COMPSCI 371 Homework 3

Group Members: Elysia Ye and Nicole Errera

Problem 0 (3 points)

Part 1: Stochastic Gradient Descent

In [1]:
```
import urllib.request
import ssl
from os import path as osp
import shutil


def retrieve(file_name, semester='fall24', homework=3):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        context = ssl._create_unverified_context()
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci371/homework/{}/{}
        url = fmt.format(semester, homework, file_name)
        with urllib.request.urlopen(url, context=context) as response:
            with open(file_name, 'wb') as file:
                shutil.copyfileobj(response, file)
        print('Downloaded file {}'.format(file_name))
```

In [2]:
```
retrieve('helpers.py', homework=2)
```

```
Using previously downloaded file helpers.py
```

In [3]:
```
from helpers import Stepper
from autograd import numpy as anp
import numpy as snp


def gradient_descent(
    f, z, alpha, min_step=1.e-6, max_iter=10000, history=False, **kwargs
):
    step = Stepper(f, z, alpha, history=history, **kwargs)
    z, fz, gz = anp.copy(z), step.fz0, step.gz0
    for k in range(max_iter):
        s, z, fz, gz = step(z, **kwargs)
        if anp.linalg.norm(s) < min_step:
            break
    step.show_history()
    return fz, z, k
```

In [4]:
```
import pickle


file_name = 'students.pkl'
retrieve(file_name)
with open(file_name, 'rb') as file:
    students = pickle.load(file)
```

```
Using previously downloaded file students.pkl
```

Problem 1.1

In [5]:
```
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(students.train.x, students.train.y)
```

In [6]:
```
w = reg.coef_
b = reg.intercept_
v = snp.concatenate((w,snp.array([b])))
print("v =", v)
```

```
v = [0.07365753 0.17634194 0.00817971 0.00533195 0.55233956]
```

In [7]:
```
def rms(v, x, y):
    quadratic_resid_risk = 0
    true_rms = 0
    for i in range(len(x)):
        cur_x = x[i]
        cur_y = y[i]
        x_adj = snp.concatenate((cur_x, snp.array([1])))
        pred_y = snp.vdot(x_adj, v)
        quadratic_resid_risk += (cur_y - pred_y)**2
        true_rms += cur_y**2
    num = (quadratic_resid_risk / len(x))**0.5
    denom = (true_rms / len(x))**0.5
    return num / denom
```

In [8]:
```
train_rms = rms(v, students.train.x, students.train.y)
test_rms = rms(v, students.test.x, students.test.y)
print("Training RMS error:", format(train_rms, '.4f'))
print("Testing RMS error:", format(test_rms, '.4f'))
```

```
Training RMS error: 0.0350
Testing RMS error: 0.0355
```

The predictor performs well. Both the training and testing RMS errors are very small, at 0.0350 and 0.0355 respectively. The predictor also generalizes well, which we know because the training and testing RMS errors are very similar. The testing RMS is within 5e-4 of the training RMS and this indicates that the predictor does well even on previously unseen data.

Problem 1.2

In [9]:
```
def risk(v, x=None, y=None, indices=None):
    sum = 0
    if indices is None: indices = range(len(x))
    for i in indices:
        cur_x = x[i]
        cur_y = y[i]
        x_adj = snp.concatenate((cur_x, snp.array([1])))
        pred_y = snp.vdot(x_adj, v)
        sum += (cur_y - pred_y)**2
    return sum / len(indices)
```

In [10]:
```
untrained_v = anp.array([1,2,3,4,5])

risk_all = risk(untrained_v, x=students.train.x, y=students.train.y)
risk_first_hundred = risk(untrained_v, x=students.train.x, y = students.trai
risk_last_hundred = risk(untrained_v, x=students.train.x, y = students.trair

print('risk for all samples:', format(risk_all, '.4f'))
print('risk for first 100 samples:', format(risk_first_hundred, '.4f'))
print('risk for last 100 samples:', format(risk_last_hundred, '.4f'))
```

```
risk for all samples: 50.1153
risk for first 100 samples: 44.3617
risk for last 100 samples: 53.0025
```

Problem 1.3

In [11]:
```
x = snp.array(students.train.x)
y = snp.array(students.train.y)

def sgd(
    h, v, alpha=1.e-3, x=x, y=y, batch_size=None,
    min_step=1.e-6, max_epochs=5000, history=True
):
    n = len(x)
    if batch_size is None:
        batch_size = n
    step = Stepper(h, v, alpha, history = history, x=x, y=y)
    not_min = True
    rng = snp.random.Generator(snp.random.PCG64())
    for k in range(max_epochs):
        perm_indices = rng.permutation(n)
        start_idx = 0
        end_idx = batch_size
        while start_idx < n and not_min:
            prev_v = v
            cur_indices = perm_indices[start_idx:min(end_idx, n)]
            start_idx = end_idx
            end_idx += batch_size
            v = step(v, indices = cur_indices, x=x, y=y)[1]
            diff = prev_v - v
            if snp.linalg.norm(diff) < min_step:
                not_min = False
        if not not_min:
```

```
                break
        step.show_history()
        return h(v, x = x, y = y), v, k
```

In [13]:
```
init_v = anp.zeros(5)
opt_risk, v, num_epochs = sgd(risk, init_v, x = x, y = y)

train_rms = rms(v, students.train.x, students.train.y)
test_rms = rms(v, students.test.x, students.test.y)

print('number of epochs:', format(num_epochs, '.4f'))
print('v =', v)
print("Training RMS error:", format(train_rms, '.4f'))
print("Testing RMS error:", format(test_rms, '.4f'))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[13], line 2
      1 init_v = anp.zeros(5)
----> 2 opt_risk, v, num_epochs = sgd(risk, init_v, x = x, y = y)
      4 train_rms = rms(v, students.train.x, students.train.y)
      5 test_rms = rms(v, students.test.x, students.test.y)

Cell In[11], line 23, in sgd(h, v, alpha, x, y, batch_size, min_step, max_ep
ochs, history)
     21 start_idx = end_idx
     22 end_idx += batch_size
---> 23 v = step(v, indices = cur_indices, x=x, y=y)[1]
     24 diff = prev_v - v
     25 if snp.linalg.norm(diff) < min_step:

File ~/Desktop/Duke/Fall24/CS371/Homeworks/hw3/helpers.py:47, in Stepper.__c
all__(self, z_old, **kwargs)
     46 def __call__(self, z_old, **kwargs):
---> 47     gz = self.g(z_old, **kwargs)
     48     s = - self.alpha * gz
     49     z = z_old + s

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/wr
ap_util.py:20, in unary_to_nary.<locals>.nary_operator.<locals>.nary_f(*arg
s, **kwargs)
     18 else:
     19     x = tuple(args[i] for i in argnum)
---> 20 return unary_operator(unary_f, x, *nary_op_args, **nary_op_kwargs)

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/di
fferential_operators.py:28, in grad(fun, x)
     21 @unary_to_nary
     22 def grad(fun, x):
     23     """
     24     Returns a function which computes the gradient of `fun` with res
pect to
     25     positional argument number `argnum`. The returned function takes
the same
     26     arguments as `fun`, but returns the gradient instead. The functi
on `fun`
     27     should be scalar-valued. The gradient has the same type as the a
rgument."""
---> 28     vjp, ans = _make_vjp(fun, x)
     29     if not vspace(ans).size == 1:
     30         raise TypeError("Grad only applies to real scalar-output fun
ctions. "
     31                         "Try jacobian, elementwise_grad or holomorph
ic_grad.")

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/co
re.py:10, in make_vjp(fun, x)
      8 def make_vjp(fun, x):
      9     start_node = VJPNode.new_root()
---> 10     end_value, end_node =  trace(start_node, fun, x)
     11     if end_node is None:
```

```
     12         def vjp(g): return vspace(x).zeros()

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/tr
acer.py:10, in trace(start_node, fun, x)
      8 with trace_stack.new_trace() as t:
      9     start_box = new_box(x, t, start_node)
---> 10     end_box = fun(start_box)
     11     if isbox(end_box) and end_box._trace == start_box._trace:
     12         return end_box._value, end_box._node

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/wr
ap_util.py:15, in unary_to_nary.<locals>.nary_operator.<locals>.nary_f.<loca
ls>.unary_f(x)
     13 else:
     14     subargs = subvals(args, zip(argnum, x))
---> 15 return fun(*subargs, **kwargs)

Cell In[9], line 8, in risk(v, x, y, indices)
      6     cur_y = y[i]
      7     x_adj = snp.concatenate((cur_x, snp.array([1])))
----> 8     pred_y = snp.vdot(x_adj, v)
      9     sum += (cur_y - pred_y)**2
     10 return sum / len(indices)

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/nu
mpy/numpy_boxes.py:26, in ArrayBox.__add__(self, other)
---> 26 def __add__(self, other): return anp.add( self, other)

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/tr
acer.py:44, in primitive.<locals>.f_wrapped(*args, **kwargs)
     42 parents = tuple(box._node for _     , box in boxed_args)
     43 argnums = tuple(argnum    for argnum, _   in boxed_args)
---> 44 ans = f_wrapped(*argvals, **kwargs)
     45 node = node_constructor(ans, f_wrapped, argvals, kwargs, argnums, pa
rents)
     46 return new_box(ans, trace, node)

File /opt/anaconda3/envs/compsci371/lib/python3.12/site-packages/autograd/tr
acer.py:35, in primitive.<locals>.f_wrapped(*args, **kwargs)
     31 def primitive(f_raw):
     32     """
     33     Wraps a function so that its gradient can be specified and its i
nvocation
     34     can be recorded. For examples, see the docs."""
---> 35     @wraps(f_raw)
     36     def f_wrapped(*args, **kwargs):
     37         boxed_args, trace, node_constructor = find_top_boxed_args(ar
gs)
     38         if boxed_args:

KeyboardInterrupt:
```

Problem 1.4 (Exam Style)

With this setting of batch_size, the SGD method is the same as regular gradient descent. Each epoch of this SGD is equivalent to one iteration of regular gradient descent.

Problem 1.5

In [ ]:
```
init_v = anp.zeros(5)
opt_risk, v, num_epochs = sgd(risk, init_v, x = x, y = y, batch_size=100)

train_rms = rms(v, students.train.x, students.train.y)
test_rms = rms(v, students.test.x, students.test.y)

print('number of epochs:', format(num_epochs, '.4f'))
print('v =', v)
print("Training RMS error:", format(train_rms, '.4f'))
print("Testing RMS error:", format(test_rms, '.4f'))
```

Problem 1.6 (Exam Style)

Problem 1.3 required computing fewer scalar derivatives.

Problem 1.7 (Exam Style)

Part 2: Linear Score-Based Classifiers

Problem 2.1 (Exam Style)

The decision boundary can be found when the scores given by the two classifiers equal to each other, meaning $s_1(x) = s_2(x) \rightarrow log(2 + 3x_1 - x_2) = log(1 - x_1 - 4x_2)$.

$log(2 + 3x_1 - x_2) = log(1 - x_1 - 4x_2)$

$2 + 3x_1 - x_2 = 1 - x_1 - 4x_2$

$4x_1 + 3x_2 + 1 = 0$, where in the required form, $b = 1$, $w_1 = 4$, and $w_2 = 3$.

This translates to the decision boundary of this classifier being $1 + 4x_1 + 3x_2 = 0$.

Problem 2.2 (Exam Style)

The decision boundary found in part 2.1 is $1 + 4x_1 + 3x_2 = 0$.

It is possible to replace $s_2(x)$ with the new $s_2'(x)$ with the new $s_1'(x)$. In this case, $s_2'(x)$ would be $s_2'(x) = (1 - x_1 - 4x_2)^3$.

Verification: $s_1'(x) = s_2'(x) \rightarrow (3 + 4x_1 + 2x_2)^3 = (1 - x_1 - 4x_2)^3$. Since the functions on both sides are monotonic, we can calculate the decision boundary using $3 + 4x_1 + 2x_2 = 1 - x_1 - 4x_2$, which allows us to reach the decision boundary found in part 2.1, which is $1 + 4x_1 + 3x_2 = 0$.

Problem 2.3 (Exam Style)

A most ambiguous point $x^* = (x_1^*, x_2^*)$ would be a point where $s_1(x) = s_2(x) = s_3(x)$.

$s_1(x) = s_2(x) = s_3(x) \rightarrow \arctan(7 + 3x_1) = \arctan(3 + 3x_1 - 2x_2) = \arctan(4 + 2x$
.

Starting by setting $s_1(x) = s_2(x) \rightarrow \arctan(7 + 3x_1) = \arctan(3 + 3x_1 - 2x_2)$, we get:

$7 + 3x_1 = 3 + 3x_1 - 2x_2 \rightarrow 2x_2 = -4$

$x_2 = -2$

Then setting
$s_1(x) = s_3(x) \rightarrow \arctan(7 + 3x_1) = \arctan(4 + 2x_1) \rightarrow 7 + 3x_1 = 4 + 2x_1$, we get:

$x_1 = -3$

Therefore we now have the most ambiguous point as $(x_1^*, x_2^*) = (-3, -2)$. Verifying that this point indeed yields the result of $s_1(x) = s_2(x) = s_3(x)$:

$s_1(x) = \arctan(7 + 3(-3)) = \arctan(-2)$

$s_2(x) = \arctan(3 + 3(-3) - 2(-2)) = \arctan(-2)$

$s_3(x) = \arctan(4 + 2(-3)) = \arctan(-2)$

This point x^* is verified.

Problem 2.4

In [21]:
```
import pickle

file_name = 'classifiers.pkl'
retrieve(file_name)
with open(file_name, 'rb') as file:
    classifiers = pickle.load(file)
```

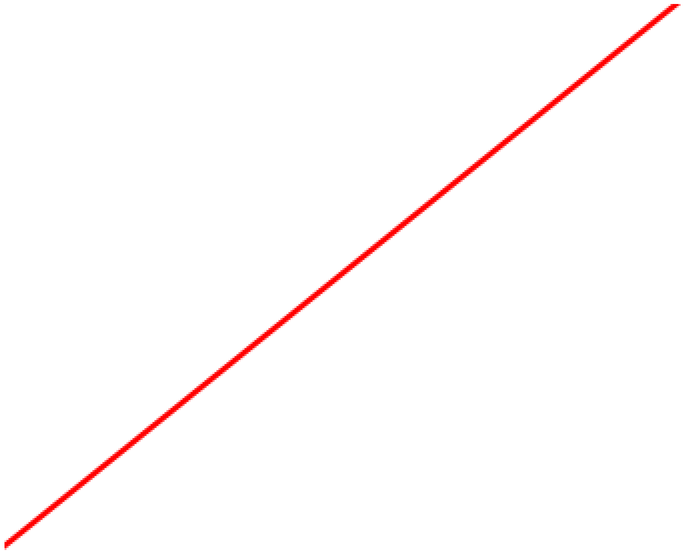
```
Using previously downloaded file classifiers.pkl
```

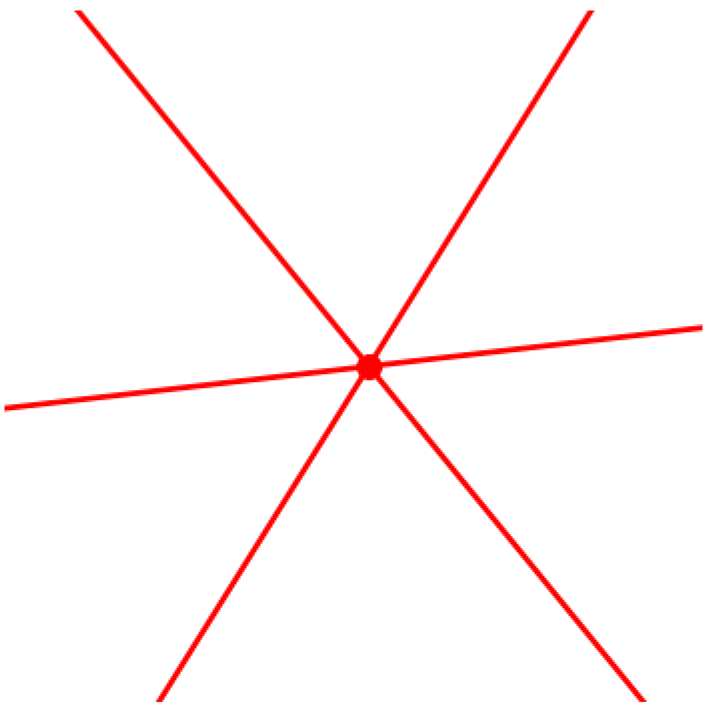
In [22]:
```
import numpy as np
import matplotlib.pyplot as plt
```

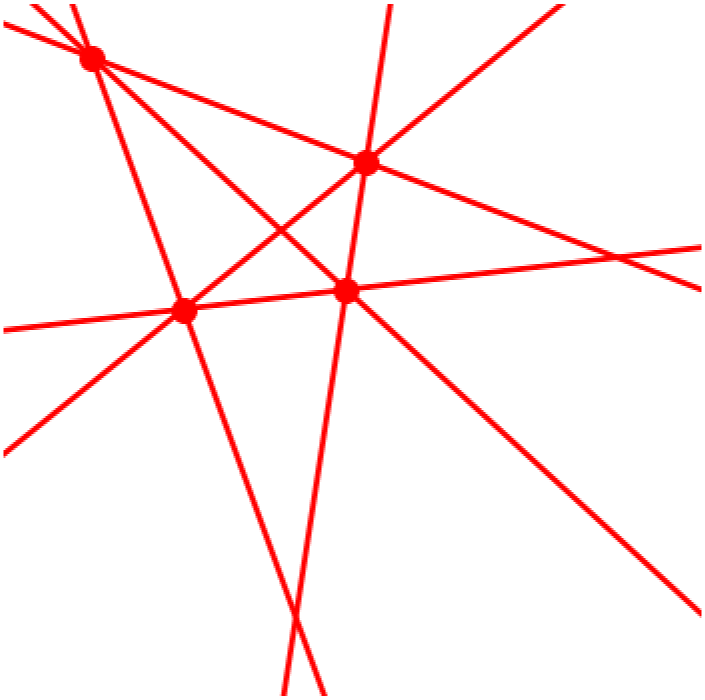
```python
def pairwise_boundaries(v):
    b = {}
    for j in range(v.shape[0]-1):
        for k in range(j+1,v.shape[0]):
            b[j,k] = (v[j]-v[k])
    return b

def triple_points(b, k):
    t = {}
    keys = list(b.keys())
    for (i, j) in keys:
        for c in range(j+1, k):
            if (j, c) in keys and (i, c) in keys:
                A = np.array([b[(i, j)], b[(j, c)]])
                point = np.linalg.solve(A[:, 1:], -A[:, 0])
                t[(i, j, c)] = point.flatten()
    return t

def show_lines(b, t, color='red'):
    ax = plt.gca()
    x_min, x_max = ax.get_xlim()
    for (i, j), params in b.items():
        x_vals = np.linspace(x_min, x_max, 300)
        y_vals = (-params[0] - params[1] * x_vals) / params[2]
        plt.plot(x_vals, y_vals, color=color, lw=2)
    for point in t.values():
        plt.scatter(*point, color=color, s=70, zorder=5)
```

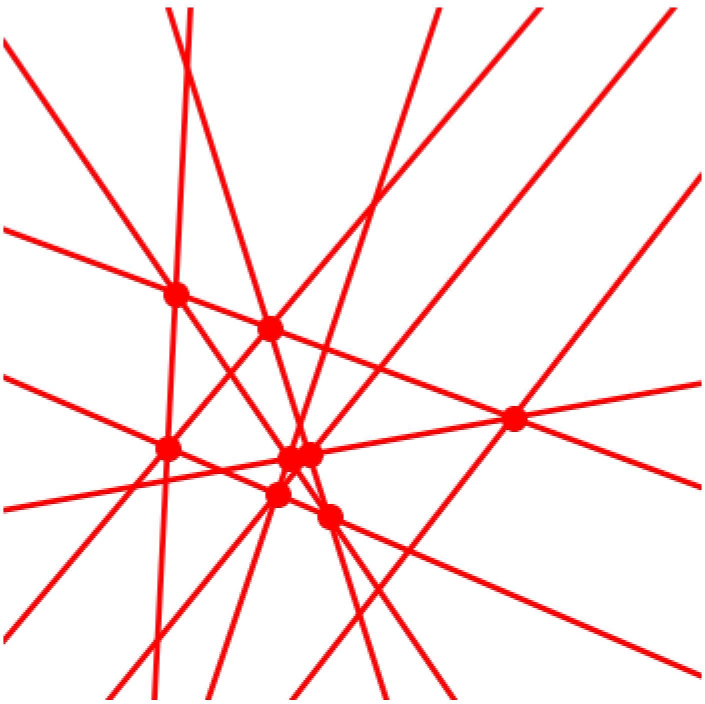
In [23]:

```python
for i, classifier in enumerate(classifiers):
        plt.figure(figsize=(4.5, 4.5))
        plt.xlim(classifier['box'][0], classifier['box'][1])
        plt.ylim(classifier['box'][2], classifier['box'][3])

        b = pairwise_boundaries(classifier['parameters'])
        t = triple_points(b, k=classifier['parameters'].shape[0])
        show_lines(b, t, color='red')
        plt.axis('off')
        plt.show()
```





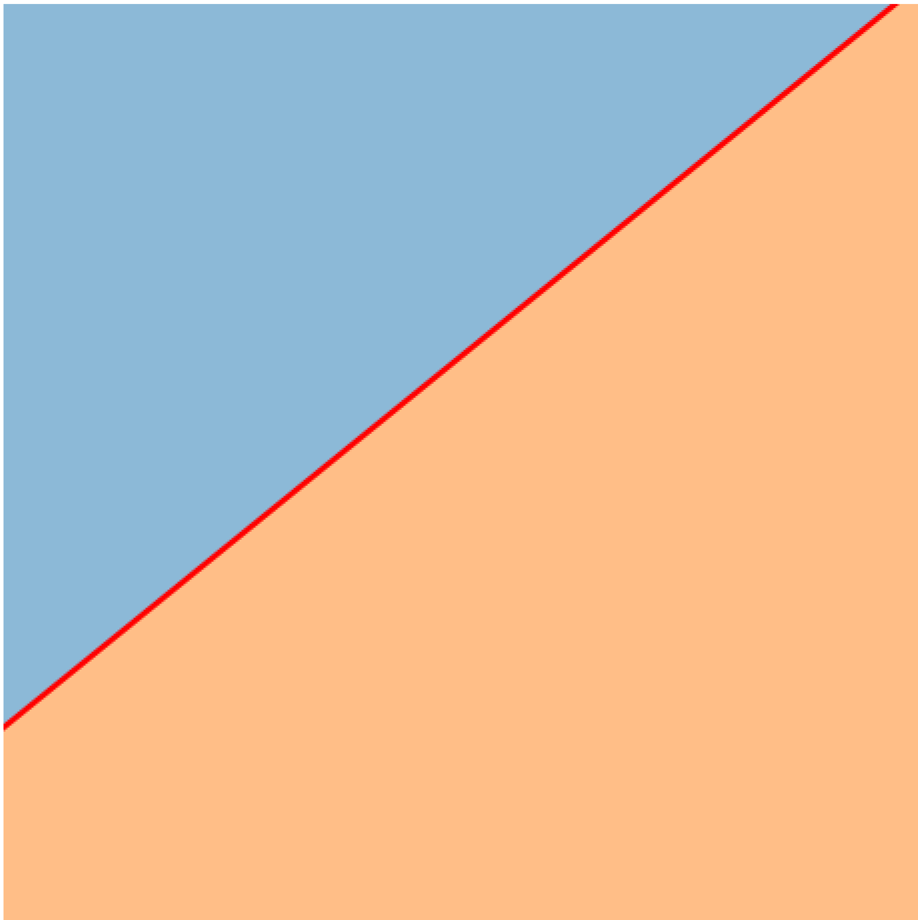


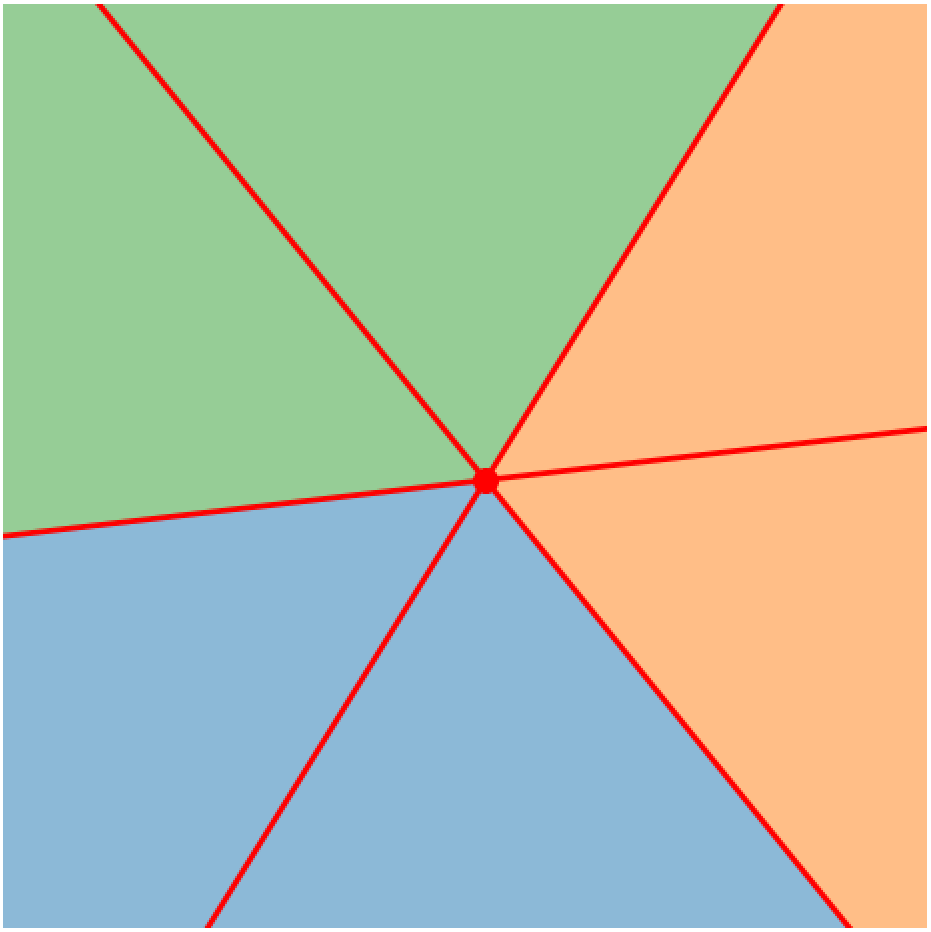


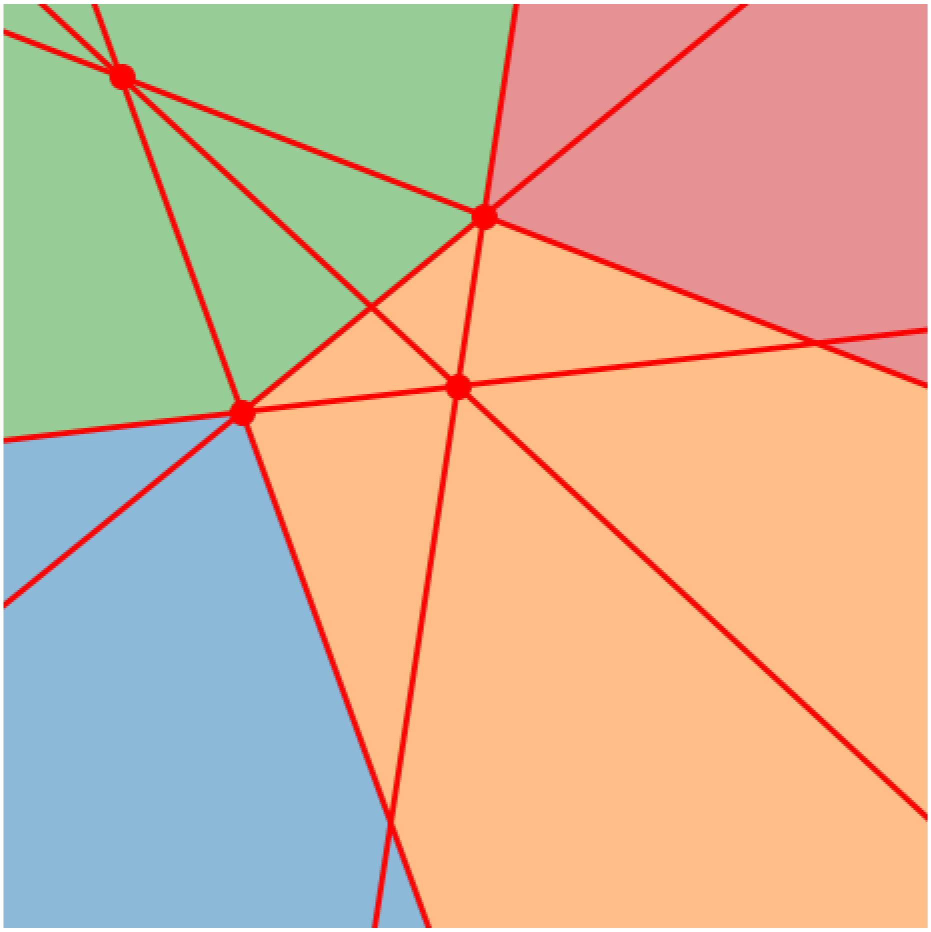
Problem 2.5

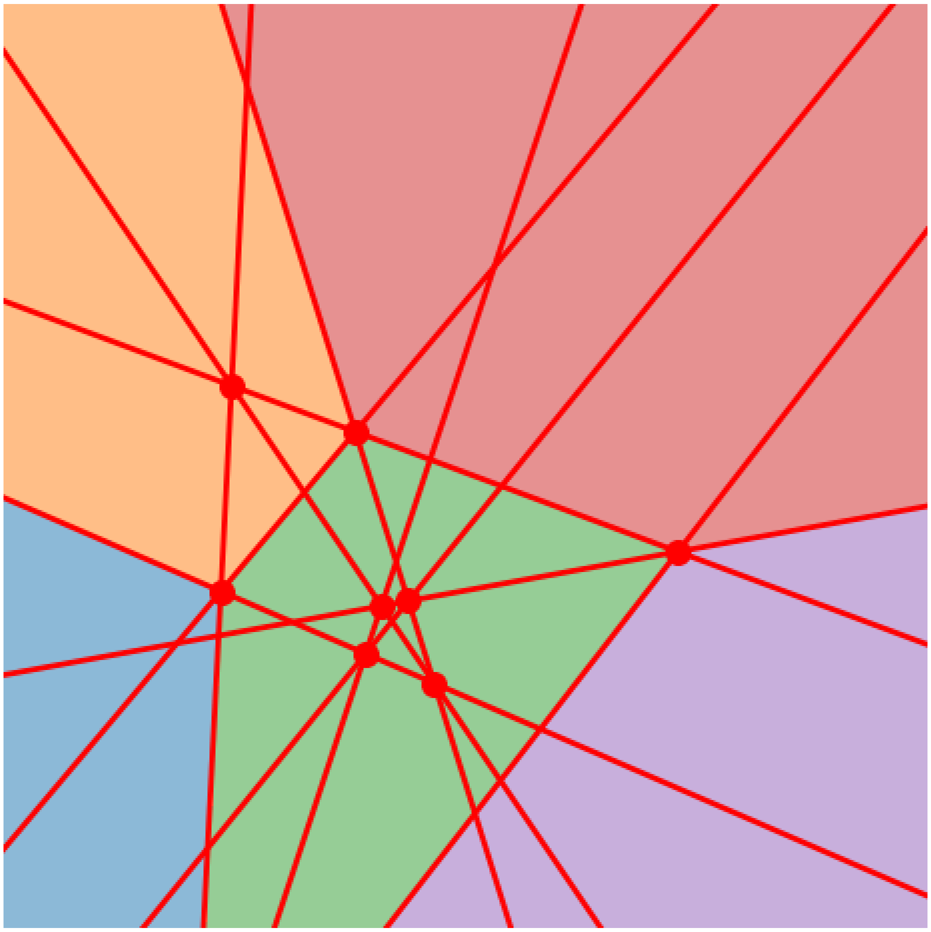
In [24]:
```
from matplotlib import colors
import numpy as np
import matplotlib.pyplot as plt

def decision_regions(v, box, g):
    x_min, x_max, y_min, y_max = box
    x = np.linspace(x_min, x_max, g)
    y = np.linspace(y_min, y_max, g)
```

```
    points = np.array([[xi, yi] for yi in y for xi in x])

    K = len(v)
    decisions = np.zeros((points.shape[0], K))

    for i in range(K):
        b, x_0, x_1 = v[i]
        decisions[:, i] = b + x_0 * points[:, 0] + x_1 * points[:, 1]

    labels = np.argmax(decisions, axis=1)
    r = labels.reshape(g, g)[::-1]
    return r

def show_regions(r, b, t, box, region_colors, line_color='red'):
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_xlim(box[0], box[1])
    ax.set_ylim(box[2], box[3])
    ax.set_aspect('equal')

    cmap = colors.ListedColormap(region_colors)

    ax.imshow(
        r, extent=box, origin='upper',
        vmin=0, vmax=len(region_colors)-1, alpha=0.5,
        cmap=cmap
    )
    show_lines(b, t, color=line_color)
    ax.axis('off')
    plt.show()

for i in range(4):
    classifier = classifiers[i]
    box = classifier['box']
    b = pairwise_boundaries(classifier['parameters'])
    t = triple_points(b, len(classifier['parameters']))
    r = decision_regions(classifier['parameters'], box=box, g=301)
    show_regions(r=r, b=b, t=t, box=box, region_colors=classifier['colors'],
```









Part 3: Linear Classification of Handwritten Digits

In [18]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from types import SimpleNamespace

digits = datasets.load_digits()
x_train, x_test, y_train, y_test = train_test_split(
    digits.data, digits.target, train_size=900, shuffle=False
)
mnist = SimpleNamespace(
    train=SimpleNamespace(x=x_train, y=y_train),
    test=SimpleNamespace(x=x_test, y=y_test)
)
```

Problem 3.1

In [19]:
```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(max_iter=1000)
classifier.fit(mnist.train.x, mnist.train.y)
train_acc = classifier.score(mnist.train.x, mnist.train.y)
test_acc = classifier.score(mnist.test.x, mnist.test.y)
```

```python
print("Training accuracy:", format(train_acc,'.3f'))
print("Testing accuracy:", format(test_acc, '.3f'))
```

```
Training accuracy: 1.000
Testing accuracy: 0.928
```

Problem 3.2 (Exam Style)

1. d = 64
2. N = 900
3. K = 10
4. m = 650
5. b = 45
6. n_{GD} = 650
7. n_{SGD} = 1950