In [26]:

```
from matplotlib import pyplot as plt
import numpy as np
import math
import time

GAS_COMPENSATION_REDUCTION = 0.995 # -> 0.5%

POPULATION_SIZE = 100 # total num
STEP_TIME = 1 # in seconds

DEBT_MEAN = 55
DEBT_SIGMA = 30

ICR_MEAN = 2
ICR_SIGMA = 0.45

STABILITY_POOL_DEPOSITS = 200 # total amount deposited into SP

BAD_TROVE_ICR = 0.8
BAD_TROVE_SIZE = 0.03 # Fraction of the total debt

#%matplotlib inline # it doesn't work with interactive animatio
n!
%matplotlib notebook

def draw(ax, points1, points2, points3, TCR, stability_pool, axe
s_size):
    ax.clear()

    ax.set_title("Troves")
    ax.set_xlabel("Debt")
    ax.set_ylabel("Collateral")
    ax.set_xlim (0, axes_size)
    ax.set_ylim (0, axes_size)

    # ICR line delimiters
    x = np.arange(0, axes_size, 1)
    y1 = x
    y2 = 1.1 * x
    ax.plot(x,y1, c='#ff0000', label='100%')
    ax.plot(x,y2, c='#0000ff', label='110%')
    y3 = TCR * x
    tcr_percentage = 100 * TCR
    ax.plot(x,y3, c='#00ff00', label='TCR: %u%%' % tcr_percentag
e)

    # Troves
    if len(points1) > 0:
        p1 = ax.scatter(points1[:,0], points1[:,1], c='#ff0000',
label='%u' % len(points1))
    if len(points2) > 0:
        p2 = ax.scatter(points2[:,0], points2[:,1], c='#0000ff',
label='%u' % len(points2))
    if len(points3) > 0:
        p3 = ax.scatter(points3[:,0], points3[:,1], c='#00ff00',
label='%u' % len(points3))
```

```
    # Stability pool
    ax.barh(0, stability_pool, height=5, color='#22BBBB', label=
'SP: %u' % stability_pool)

    # Legend
    ax.legend(loc='upper left', frameon=False)

def redistribute(points, total_coll, bad_trove_debt, bad_trove_c
oll):
    for t in points:
        t[0] += bad_trove_debt * t[1] / total_coll
        t[1] += bad_trove_coll * t[1] * GAS_COMPENSATION_REDUCTI
ON / total_coll # Discount 0.5% for gas compensation
        t[2] = t[1] / t[0]
    return points

def shift_points(points1, points2, points3, shifted_to_1, shifte
d_to_2):
    if len(points3) > 0:
        # Points are ordered by ICR, so once we find one correct
we are done
        while points3[0][2] < 1.1:
            point_to_move, points3 = points3[0], points3[1:]
            points2 = np.append(points2, [point_to_move], axis=
0)
            shifted_to_2 += 1
    if len(points2) > 0:
        # Points are ordered by ICR, so once we find one correct
we are done
        while points2[0][2] < 1:
            point_to_move, points2 = points2[0], points2[1:]
            points1 = np.append(points1, [point_to_move], axis=
0)
            shifted_to_1 += 1
    return points1, points2, points3, shifted_to_1, shifted_to_2

def liquidate_trove(points1, points2, points3, bad_trove_debt, b
ad_trove_coll, stability_pool, shifted_to_1, shifted_to_2, alway
s_offset = False):
    # If profitable, redistribute against Stability Pool
    if stability_pool > 0 and (bad_trove_coll / bad_trove_debt >
= 1 or always_offset):
        if stability_pool > bad_trove_debt:
            stability_pool -= bad_trove_debt
            bad_trove_debt = 0
            bad_trove_coll = 0
        else:
            new_bad_trove_debt = bad_trove_debt - stability_pool
            bad_trove_coll = bad_trove_coll * new_bad_trove_debt
/ bad_trove_debt
            bad_trove_debt = new_bad_trove_debt
            stability_pool = 0

    # Redistribute remaining
    total_debt = sum(points1[:,0]) + sum(points2[:,0]) + sum(poi
nts3[:,0])
    total_coll = sum(points1[:,1]) + sum(points2[:,1]) + sum(poi
nts3[:,1])
    if bad_trove_debt > 0:
        points1 = redistribute(points1, total_coll, bad_trove_de
```

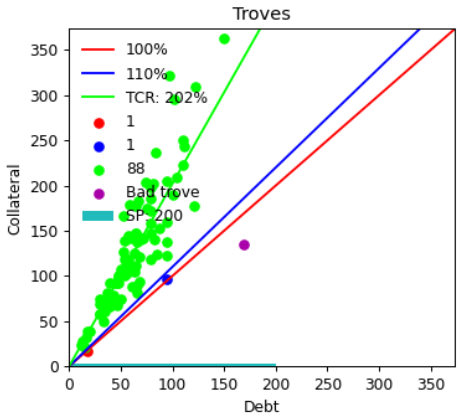
```
bt, bad_trove_coll)
        points2 = redistribute(points2, total_coll, bad_trove_de
bt, bad_trove_coll)
        points3 = redistribute(points3, total_coll, bad_trove_de
bt, bad_trove_coll)

        # Check if any trove needs to be moved to the previous a
rray due to the avalanche effect
        points1, points2, points3, shifted_to_1, shifted_to_2 =
\
            shift_points(points1, points2, points3, shifted_to_
1, shifted_to_2)

    newTCR = total_coll / total_debt
    return points1, points2, points3, newTCR, stability_pool, sh
ifted_to_1, shifted_to_2

def generate_table_data(points1_1, points2_1, points3_1, points1
_0, points2_0, points3_0, shifted_to_1_1, shifted_to_2_1, shifte
d_to_1_0, shifted_to_2_0):
    total_debt_1 = sum(points1_1[:,0]) + sum(points2_1[:,0]) + s
um(points3_1[:,0])
    total_coll_1 = sum(points1_1[:,1]) + sum(points2_1[:,1]) + s
um(points3_1[:,1])
    tcr_1 = 100 * total_coll_1 / total_debt_1
    total_debt_0 = sum(points1_0[:,0]) + sum(points2_0[:,0]) + s
um(points3_0[:,0])
    total_coll_0 = sum(points1_0[:,1]) + sum(points2_0[:,1]) + s
um(points3_0[:,1])
    tcr_0 = 100 * total_coll_0 / total_debt_0
    return [
        ['', 'New method', 'Old method'],
        ['Number of troves', len(points3_1) + len(points2_1), le
n(points3_0) + len(points2_0)],
        ['Healthy troves (> 110%)', len(points3_1), len(points3_
0)],
        ['Total Collateral', '%.2f' % total_coll_1, '%.2f' % tot
al_coll_0],
        ['Total Debt', '%.2f' % total_debt_1, '%.2f' % total_deb
t_0],
        ['TCR', '%.2f' % tcr_1, '%.2f' % tcr_0],
        ['Troves drawdowned below 110%', shifted_to_2_1, shifted
_to_2_0],
        ['Troves drawdowned below 100%', shifted_to_1_1, shifted
_to_1_0],
    ]

debts = list(filter(lambda x: x > 10, np.random.normal(DEBT_MEA
N, DEBT_SIGMA, POPULATION_SIZE)))

icrs = np.random.normal(loc=ICR_MEAN, scale=ICR_SIGMA, size=len
(debts))
# TODO: filter icrs, and re-adjust debts

colls = [a * b for a, b in zip(debts, icrs)]

points = np.array([debts, colls, icrs]).transpose()
# split
points1 = points[points[:, 2] < 1]
points2 = points[(points[:, 2] >= 1) & (points[:, 2] < 1.1)]
```

```
points3 = points[points[:, 2] >= 1.1]
# sort
points1_1 = points1[points1[:,2].argsort()]
points2_1 = points2[points2[:,2].argsort()]
points3_1 = points3[points3[:,2].argsort()]

# copy to apply the old method
points1_0 = np.copy(points1_1)
points2_0 = np.copy(points2_1)
points3_0 = np.copy(points3_1)

sum_debts = sum(debts)
BAD_TROVE_DEBT = sum_debts * BAD_TROVE_SIZE
BAD_TROVE_COLL = BAD_TROVE_DEBT * BAD_TROVE_ICR

AXES_SIZE = max(max(debts), max(colls), BAD_TROVE_DEBT) + 10

total_coll = sum(colls) + BAD_TROVE_COLL
total_debt = sum_debts + BAD_TROVE_DEBT
TCR_1 = total_coll / total_debt

TCR_0 = TCR_1

# init SPs
stability_pool_1 = STABILITY_POOL_DEPOSITS
stability_pool_0 = STABILITY_POOL_DEPOSITS

# Drawdown counters
shifted_to_1_1 = 0
shifted_to_2_1 = 0
shifted_to_1_0 = 0
shifted_to_2_0 = 0

# Start drawing


# The really important command for interactive plot updating
plt.ion()

# sizing of the plots figure sizes
fig_size = (10, 4)
fig_size_two_rows = (10, 8)

fig1 = plt.figure('Initial situation', figsize=fig_size)
ax1 = fig1.add_subplot(121)
fig1.canvas.draw()
draw(ax1, points1_1, points2_1, points3_1, TCR_1, stability_pool
_1, AXES_SIZE)

# Add “bad trove”
ax1.scatter(BAD_TROVE_DEBT, BAD_TROVE_COLL, c='#AA00AA', label='
Bad trove')
ax1.legend(loc='upper left', frameon=False)

# Add summary table
ax1_2 = fig1.add_subplot(122)
table_data_1 = [
    ['Number of troves', POPULATION_SIZE + 1],
    ['Healthy troves (> 110%)', len(points3)],
    ['Bad trove Collateral', '%.2f' % BAD_TROVE_COLL],
```

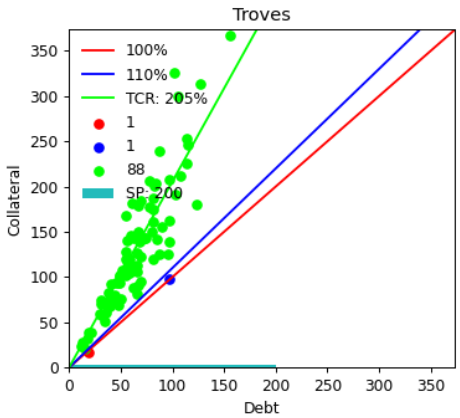
```
    ['Bad trove Debt', '%.2f' % BAD_TROVE_DEBT],
    ['Total Collateral', '%.2f' % total_coll],
    ['Total Debt', '%.2f' % total_debt],
]
ax1_2.table(cellText=table_data_1, loc='center')
ax1_2.axis('off')

fig1.canvas.draw()

#fig1.savefig('loss_evasion_frontrunning_1_0000.png')


# Liquidate initial bad trove
fig2 = plt.figure('After initial bad trove liquidation', figsize
=fig_size)

# new method
ax2_1 = fig2.add_subplot(121)
fig2.canvas.draw()
(points1_1, points2_1, points3_1, TCR_1, stability_pool_1, shift
ed_to_1_1,
 shifted_to_2_1) = \
    liquidate_trove(points1_1, points2_1, points3_1,
                    BAD_TROVE_DEBT, BAD_TROVE_COLL, stability_po
ol_1,
                    shifted_to_1_1, shifted_to_2_1)
draw(ax2_1, points1_1, points2_1, points3_1, TCR_1, stability_po
ol_1, AXES_SIZE)

# old method
ax2_0 = fig2.add_subplot(122)
fig2.canvas.draw()
(points1_0, points2_0, points3_0, TCR_0, stability_pool_0, shift
ed_to_1_0,
            shifted_to_2_0) = \
liquidate_trove(points1_0, points2_0, points3_0,
                            BAD_TROVE_DEBT, BAD_TROVE_COLL, stabi
lity_pool_0,
                            shifted_to_1_0, shifted_to_2_0, alway
s_offset=True)
draw(ax2_0, points1_0, points2_0, points3_0, TCR_0, stability_po
ol_0, AXES_SIZE)

fig2.canvas.draw()
#fig2.savefig('loss_evasion_frontrunning_2_0000.png')


# Liquidate the rest

# Liquidate remaining *unprofitable* troves (ICR < 100%)
fig3 = plt.figure('After liquidation of unprofitable troves', fi
gsize=fig_size_two_rows)
# new method
ax3_1 = fig3.add_subplot(223)
fig3.canvas.draw()
#i = 0
while len(points1_1) > 0:
    bad_trove_1, points1_1 = points1_1[0], points1_1[1:]
    points1_1, points2_1, points3_1, TCR_1, stability_pool_1, sh
ifted_to_1_1, shifted_to_2_1 = \
```

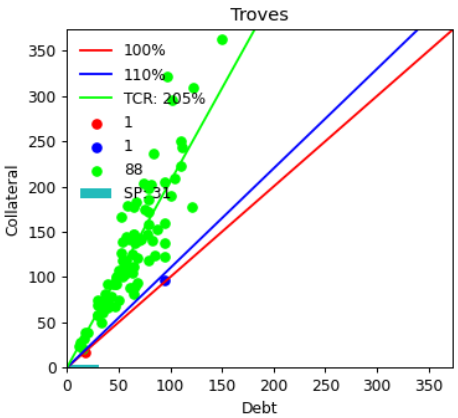
```
        liquidate_trove(points1_1, points2_1, points3_1, bad_tro
ve_1[0], bad_trove_1[1], stability_pool_1,
                        shifted_to_1_1, shifted_to_2_1)
    draw(ax3_1, points1_1, points2_1, points3_1, TCR_1, stabilit
y_pool_1, AXES_SIZE)
    fig3.canvas.draw()
    #fig3.savefig('loss_evasion_frontrunning_3_%04d.png' % i)
    #i += 1
    time.sleep(STEP_TIME)
# old method
ax3_0 = fig3.add_subplot(224)
fig3.canvas.draw()
while len(points1_0) > 0:
    bad_trove_0, points1_0 = points1_0[0], points1_0[1:]
    points1_0, points2_0, points3_0, TCR_0, stability_pool_0, sh
ifted_to_1_0, shifted_to_2_0 = \
    liquidate_trove(points1_0, points2_0, points3_0, bad_trove_0
[0], bad_trove_0[1], stability_pool_0,
                                shifted_to_1_0, shifted_to_2_0, a
lways_offset=True)
    draw(ax3_0, points1_0, points2_0, points3_0, TCR_0, stabilit
y_pool_0, AXES_SIZE)
    fig3.canvas.draw()
    time.sleep(STEP_TIME)

# Add summary table
ax3_2 = fig3.add_subplot(211)
table_data_3 = generate_table_data(points1_1, points2_1, points3
_1, points1_0, points2_0, points3_0, shifted_to_1_1, shifted_to_
2_1, shifted_to_1_0, shifted_to_2_0)
ax3_2.table(cellText=table_data_3, loc='center')
ax3_2.axis('off')

# Liquidate *profitable* troves (100 <= ICR < 110%)
#fig4, ax4 = plt.subplots()
fig4 = plt.figure('After liquidation of profitable troves', figs
ize=fig_size_two_rows)
# new method
ax4_1 = fig4.add_subplot(223)
fig4.canvas.draw()
#i = 0
while len(points2_1) > 0:
    bad_trove_1, points2_1 = points2_1[0], points2_1[1:]
    points1_1, points2_1, points3_1, TCR_1, stability_pool_1, sh
ifted_to_1_1, shifted_to_2_1 = \
        liquidate_trove(points1_1, points2_1, points3_1, bad_tro
ve_1[0], bad_trove_1[1],
                        stability_pool_1, shifted_to_1_1, shifte
d_to_2_1)
    draw(ax4_1, points1_1, points2_1, points3_1, TCR_1, stabilit
y_pool_1, AXES_SIZE)
    fig4.canvas.draw()
    #fig4.savefig('loss_evasion_frontrunning_4_%04d.png' % i)
    #i += 1
    time.sleep(STEP_TIME)
# old method
ax4_0 = fig4.add_subplot(224)
fig4.canvas.draw()
while len(points2_0) > 0:
    bad_trove_0, points2_0 = points2_0[0], points2_0[1:]
```

```
    points1_0, points2_0, points3_0, TCR_0, stability_pool_0, sh
ifted_to_1_0, shifted_to_2_0 = \
    liquidate_trove(points1_0, points2_0, points3_0, bad_trove_0
[0], bad_trove_0[1],
                    stability_pool_0, shifted_to_1_0, shifted_to
_2_0, always_offset=True)
    draw(ax4_0, points1_0, points2_0, points3_0, TCR_0, stabilit
y_pool_0, AXES_SIZE)
    fig4.canvas.draw()
    time.sleep(STEP_TIME)

# Add summary table
ax4_2 = fig4.add_subplot(211)
table_data_4 = generate_table_data(points1_1, points2_1, points3
_1, points1_0, points2_0, points3_0, shifted_to_1_1, shifted_to_
2_1, shifted_to_1_0, shifted_to_2_0)
ax4_2.table(cellText=table_data_4, loc='center')
ax4_2.axis('off')
```

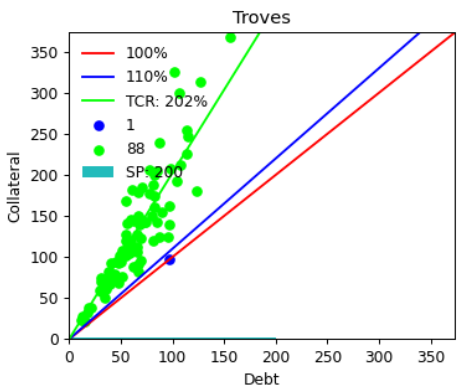


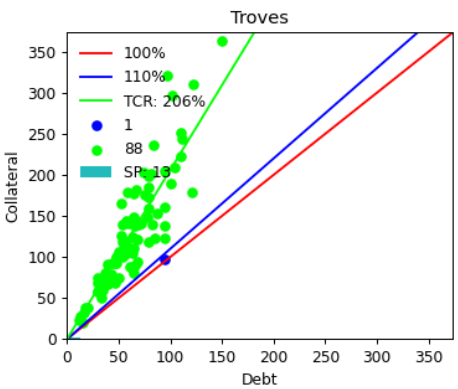
Number of troves	101
Healthy troves (> 110%)	88
Bad trove Collateral	134.89
Bad trove Debt	168.61
Total Collateral	11695.80
Total Debt	5789.07



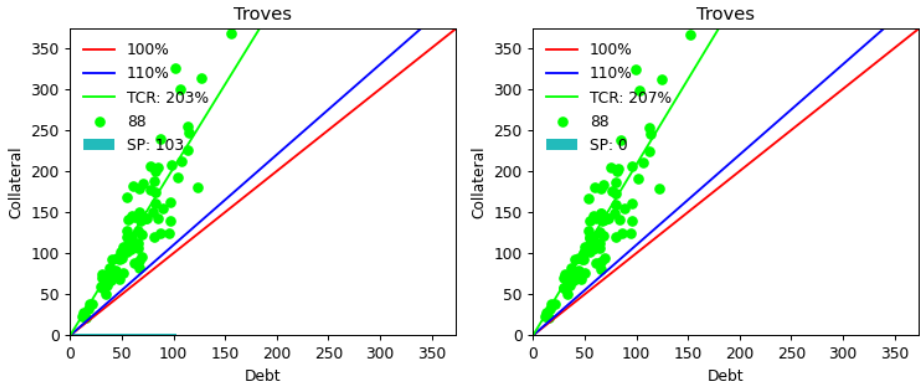


	New method	Old method
Number of troves	89	89
Healthy troves (> 110%)	88	88
Total Collateral	11695.04	11544.04
Total Debt	5789.07	5602.32
TCR	202.02	206.06
Troves drawdowned below 110%	0	0
Troves drawdowned below 100%	0	0





	New method	Old method
Number of troves	88	88
Healthy troves (> 110%)	88	88
Total Collateral	11597.08	11530.14
Total Debt	5692.46	5589.07
TCR	203.73	206.30
Troves drawdowned below 110%	0	0
Troves drawdowned below 100%	0	0



Out[26]: (0.0, 1.0, 0.0, 1.0)

In [ ]: