spin_charge_symm

April 20, 2021

[1]:

```python
import itertools
from tqdm import tqdm
from time import sleep
from math import sqrt
import multiprocessing as mp
from multiprocessing import Pool
#mp.set_start_method('spawn')

import matplotlib
from matplotlib import pyplot as plt

import numpy as np

font = {'size'   : 20}

matplotlib.rc('font', **font)
matplotlib.rcParams['text.usetex'] = True
plt.rcParams["figure.figsize"]= 7, 5
plt.rcParams['figure.dpi'] = 90
matplotlib.rcParams['lines.linewidth'] = 2
plt.rcParams['axes.grid'] = True
plt.style.use('seaborn-whitegrid')
```

Defines and evaluates denominators in the RG equations. The denominators in the RG equations are

$$d_0 = \omega - \frac{1}{2}\left(D - \mu\right) - \frac{U}{2} + \frac{K}{2}$$

$$d_1 = \omega - \frac{1}{2}\left(D - \mu\right) + \frac{U}{2} + \frac{J}{2}$$

$$d_2 = \omega - \frac{1}{2}\left(D - \mu\right) + \frac{J}{4} + \frac{K}{4}$$

[2]:

```python
def den(w, D, U, J, K):
    d0 = w - 0.5 * D - U/2 + K/2
    d1 = w - 0.5 * D + U/2 + J/2
    d2 = w - 0.5 * D + J/4 + K/4
    return d0, d1, d2
```

1 RG Equations

The RG equations for the symmetric spin-charge Anderson-Kondo are

$$\Delta U = 4|V|^2 \left[\frac{1}{\omega - \frac{1}{2}(D-\mu) + \frac{U}{2} + \frac{1}{2}J} - \frac{1}{\omega - \frac{1}{2}(D-\mu) - \frac{U}{2} + \frac{1}{2}K}\right] + \sum_{k<\Lambda_j} \frac{3}{4} \frac{K^2 - J^2}{\omega - \frac{1}{2}(D-\mu) + \frac{1}{4}J + \frac{1}{4}K}$$

$$\Delta V = \frac{VK}{16}\left(\frac{1}{\omega - \frac{1}{2}(D-\mu) - \frac{U}{2} + \frac{1}{2}K} + \frac{1}{\omega - \frac{1}{2}(D-\mu) + \frac{1}{4}J + \frac{1}{4}K}\right) - \frac{3VJ}{4}\left(\frac{1}{\omega - \frac{1}{2}(D-\mu) + \frac{U}{2} + \frac{1}{2}J} + \frac{}{\omega - \frac{1}{2}\,(}\right.$$

$$\Delta J = -J^2 \left(\omega - \frac{1}{2}(D-\mu) + \frac{1}{4}J + \frac{1}{4}K\right)^{-1}$$

$$\Delta K = -K^2 \left(\omega - \frac{1}{2}(D-\mu) + \frac{1}{4}J + \frac{1}{4}K\right)^{-1}$$

The following equation accepts the coupling values at the j^{th} step of the RG, applies the RG equations on them and returns the couplings for the $(j-1)^{th}$ step. *If any coupling changes sign, it is set to 0.*

[3]:
```
def rg(w, D, U, V, J, K):
    dens = den(w, D, U, J, K)
    deltaU = -4 * V**2 * (1/dens[0] - 1/dens[1]) - (3* (J**2 - K**2)/8) * D /␣
↪dens[2]
    deltaV = (1/16) * K * V * (1/dens[0] - 1/dens[2]) - (3/4) * J * V * (1/
↪dens[1] + 1/dens[2])
    deltaJ = - J**2 / dens[2]
    deltaK = - K**2 / dens[2]

    #n = 2*np.pi*N*sqrt(D/Df)
    n = 1
    U = 0 if (U + n*deltaU) * U <= 0 else U + n*deltaU
    V = 0 if (V + n*deltaV) * V <= 0 else V + n*deltaV
    J = 0 if (J + n*deltaJ) * J <= 0 else J + n*deltaJ
    K = 0 if (K + n*deltaK) * K <= 0 else K + n*deltaK

    return U, V, J, K
```

The following function does one complete RG for a given set of bare couplings and returns arrays of the flowing couplings.

[4]:
```
def complete_RG(w, D0, U0, V0, J0, K0):
    U = U0
    V = V0
    J = J0
    K = K0
    N = 100
    old_den = den(w, D0, U, J, K)[2]
    x, y1, y2, y3, y4, y5 = [], [], [], [], [], []
    flag = False
    count = N+1
    for D in np.linspace(D0, 0, N):
        count -= 1
        x.append(D)
        y1.append(U)
        y2.append(J)
        y3.append(K)
        y4.append(V)
        y5.append(count)
        new_den = den(w, D, U, J, K)[2]
        if old_den * new_den <= 0:
            flag = True
            return [x, y1, y2, y3, y4, flag, y5]
        old_den = new_den
        U, V, J, K = rg(w, D, U, V, J, K)
    return [x, y1, y2, y3, y4, flag, y5]
```

2 1. $V = 0$

First we will look at the simplified case of $V = 0$. Since the RG equation for V involves V, it will not flow. We need to look only at U, J and K. Depending on the value of ω, the denominator can be either positive or negative. We look at the two cases separately.

2.1 a. $\omega - \frac{\epsilon_q}{2} + \frac{1}{4}J + \frac{1}{4}K > 0$ (high ω):

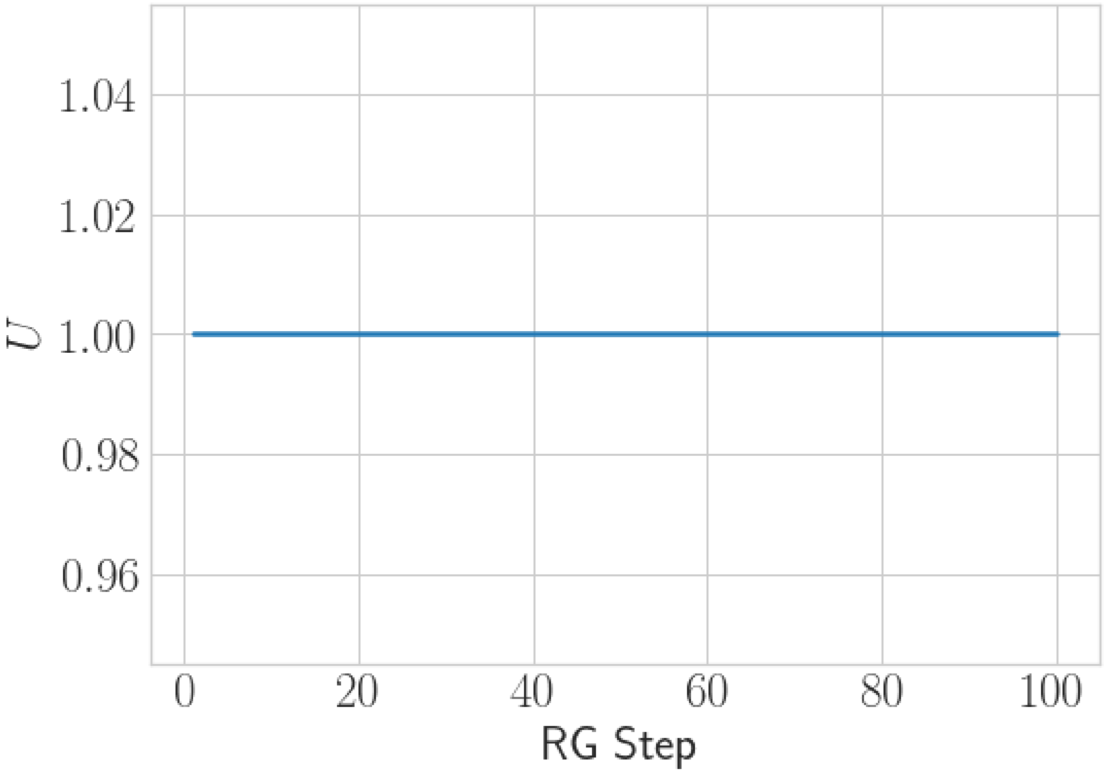
These aren't truly URG fixed points because the denominator will not converge towards zero.

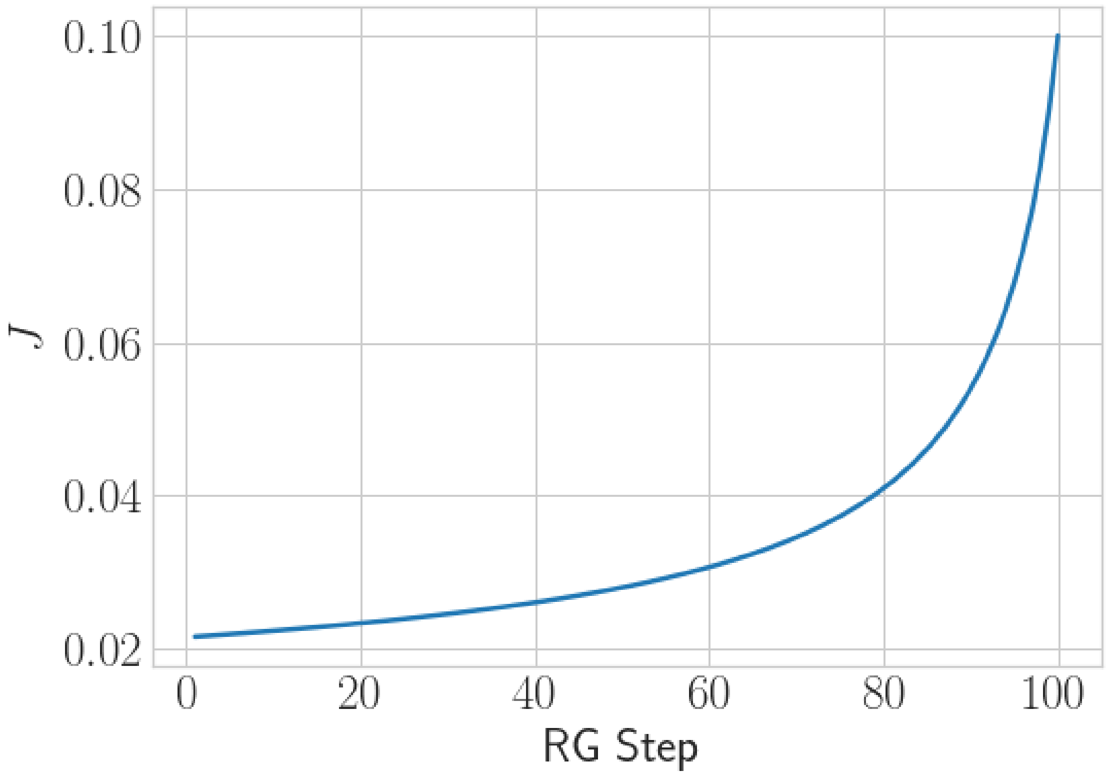
2.1.1 i. $J = K$

Since $\Delta U \propto K^2 - J^2$, U will be marginal here.

[5]:

```
w = 6
D0 = 10
U0 = 1
J0 = K0 = 0.1
V0 = 0
Df = D0/2
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
plt.ylabel(r'$U$')
plt.xlabel(r'RG Step')
plt.plot(y5, y1)
plt.show()
plt.ylabel(r'$J$')
plt.xlabel(r'RG Step')
plt.plot(y5, y2)
plt.show()
```



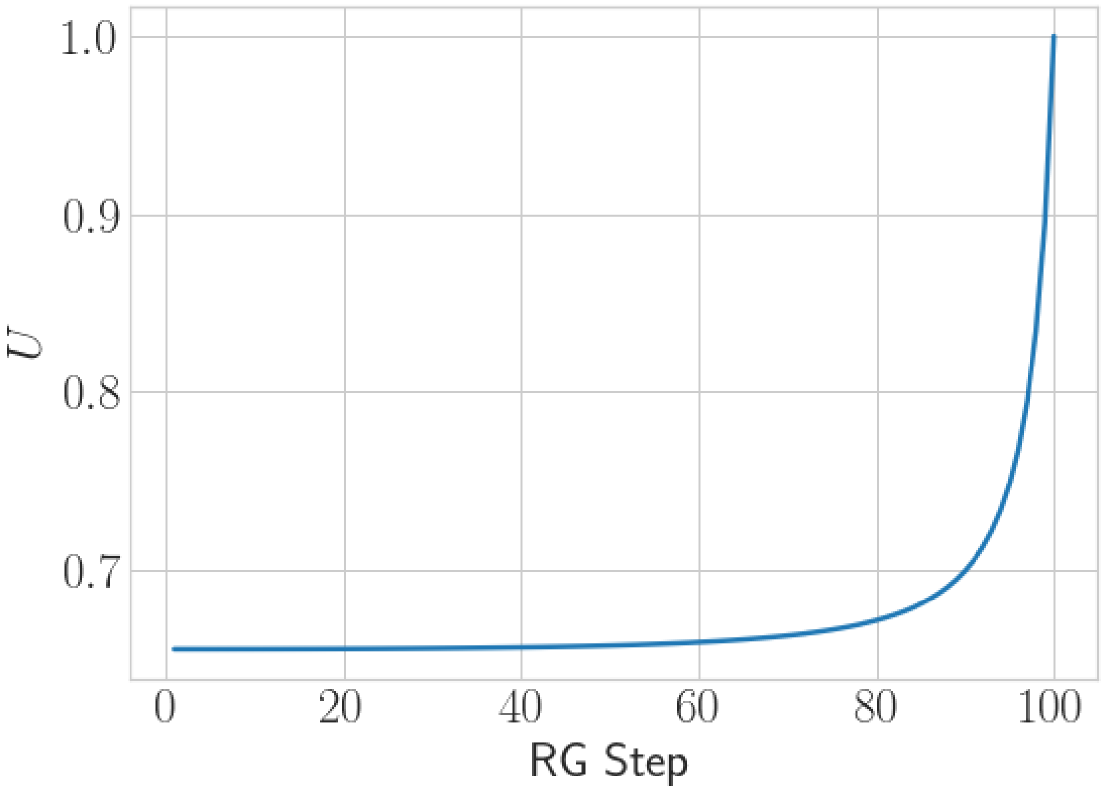


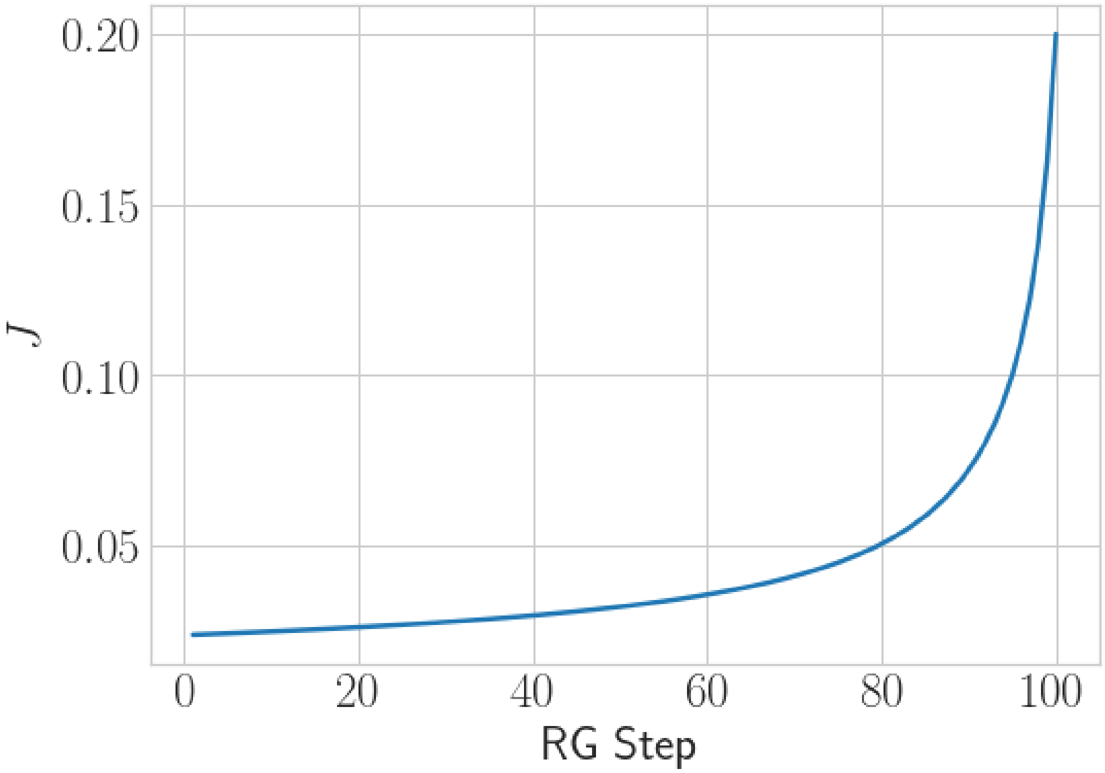
2.1.2 ii. $J > K$

Since $\Delta U \propto K^2 - J^2$, U will be irrelevant here.

[6]:

```
w = 6
D0 = 10
U0 = 1
J0 = 0.2
K0 = 0.1
V0 = 0
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
plt.ylabel(r'$U$')
plt.xlabel(r'RG Step')
plt.plot(y5, y1)
plt.show()
plt.ylabel(r'$J$')
plt.xlabel(r'RG Step')
plt.plot(y5, y2)
plt.show()
```


1.0
0.9
0.8
0.7
U
0
20
40
60
80
100
RG Step

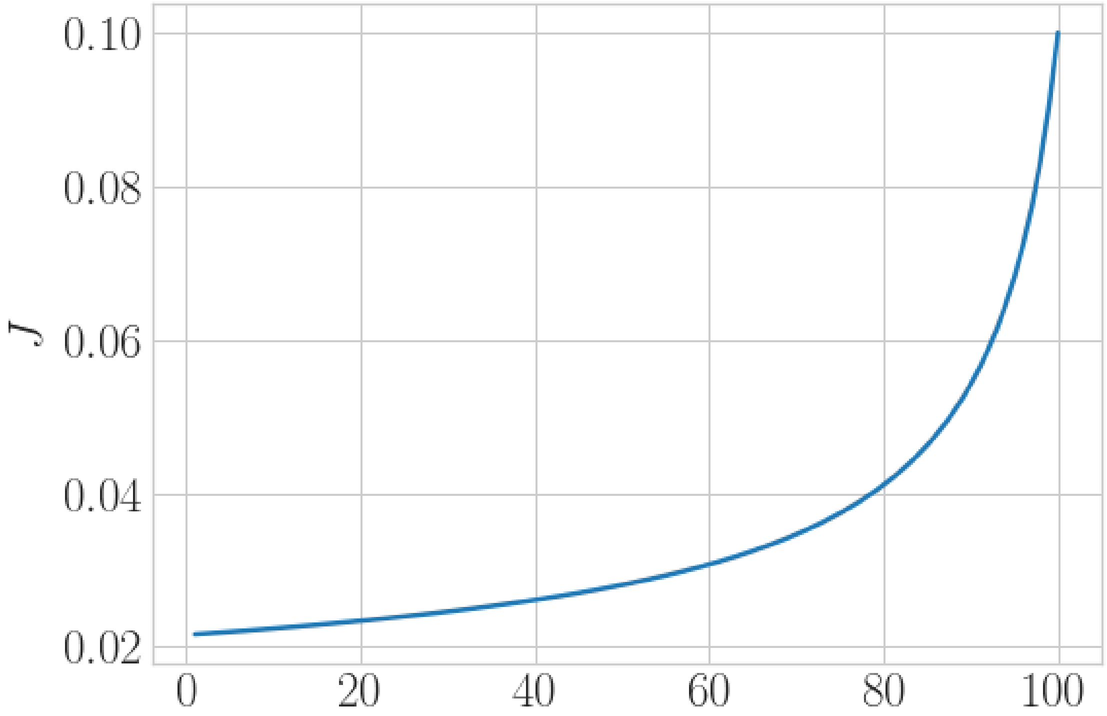

0.20
0.15
0.10
0.05
J
0
20
40
60
80
100
RG Step

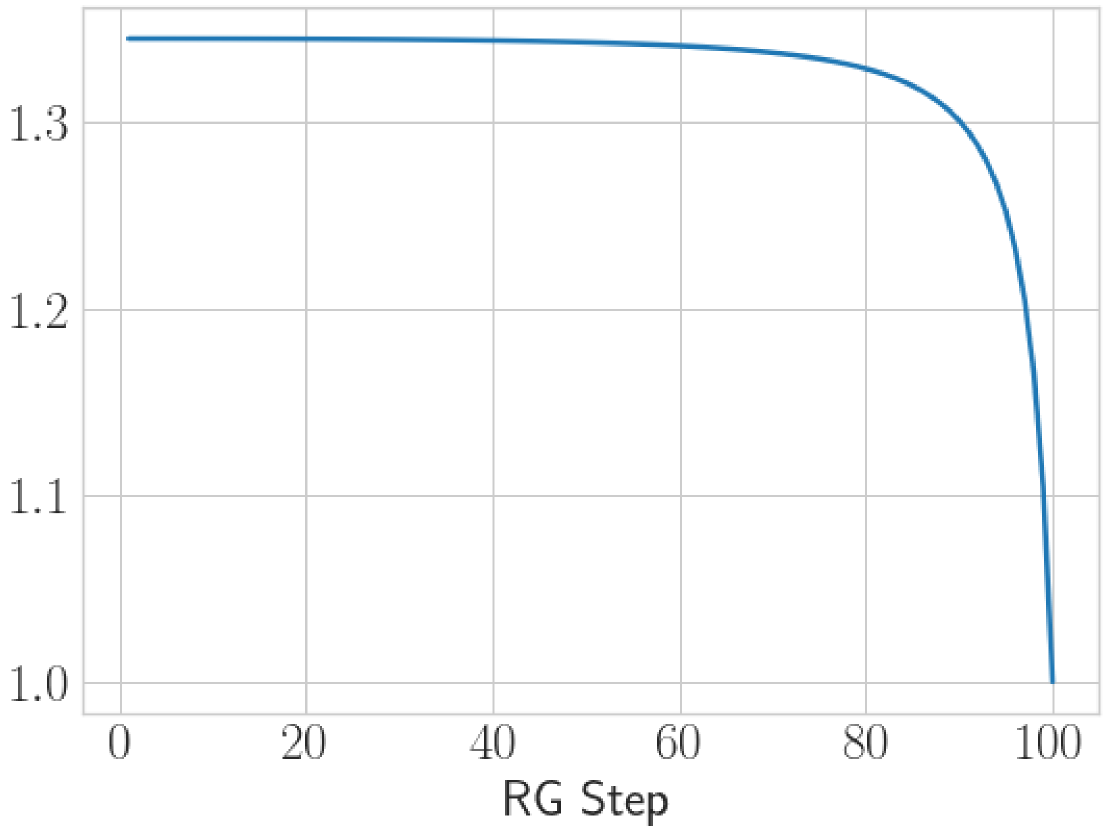
2.1.3 iii. $J < K$

Since $\Delta U \propto K^2 - J^2$, U will be relevant here.

[7]:

```
w = 6
D0 = 10
U0 = 1
J0 = 0.1
K0 = 0.2
V0 = 0
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
plt.ylabel(r'$U$')
plt.ylabel(r'$J$')
plt.plot(y5, y2)
plt.show()
plt.xlabel(r'RG Step')
plt.xlabel(r'RG Step')
plt.plot(y5, y1)
plt.show()
```





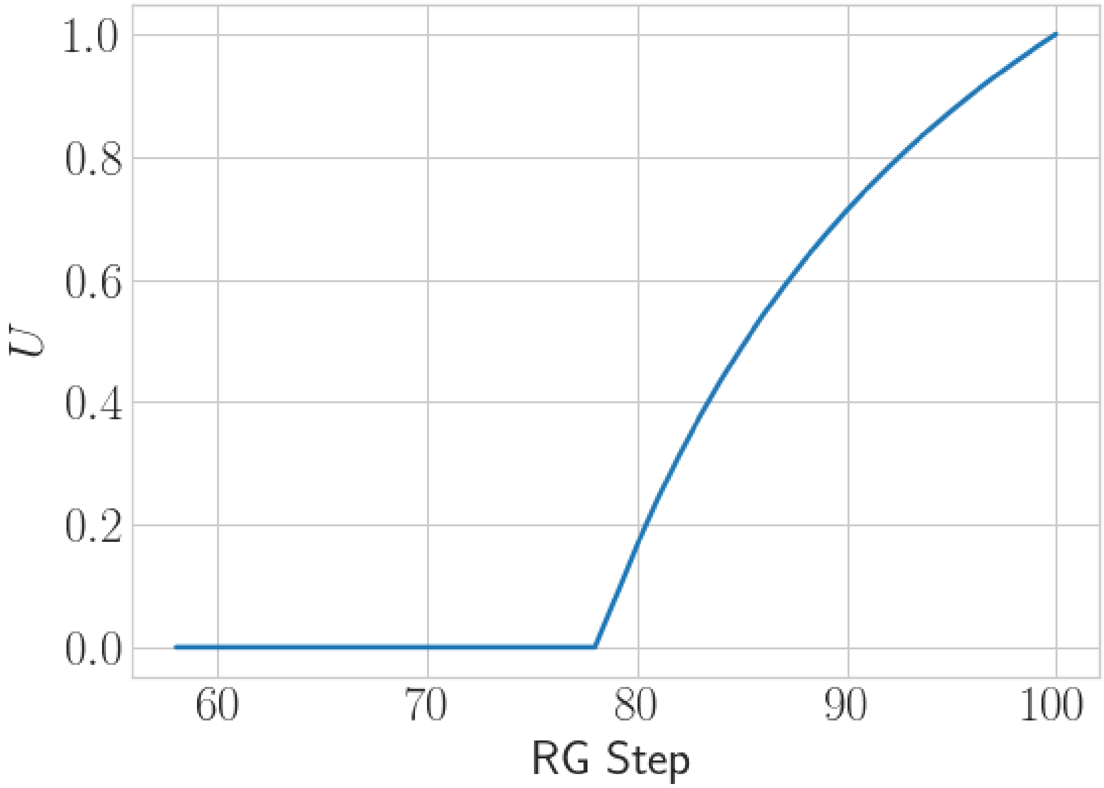
2.2 b. $\omega - \frac{\epsilon_q}{2} + \frac{1}{4}J + \frac{1}{4}K < 0$ (low ω):

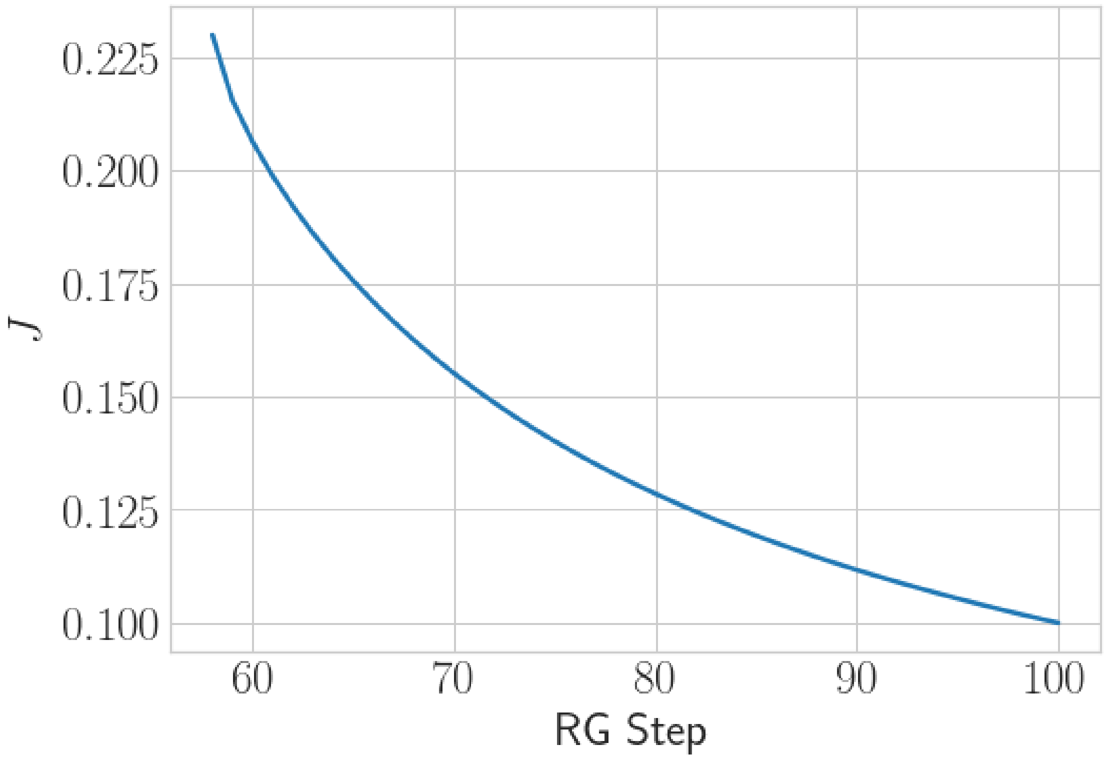
This is the regime where we achieve true strong-coupling fixed points in J, K. The signature of $K^2 - J^2$ will determine whether U is relevant or irrelevant.

2.2.1 i. $J > K$

[8]:

```
w = 0.01
D0 = 20
U0 = 1
J0 = 0.1
K0 = 0.2
V0 = 0
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
if flag == True:
    plt.ylabel(r'$U$')
    plt.plot(y5, y1)
    plt.xlabel(r'RG Step')
    plt.show()
    plt.ylabel(r'$J$')
    plt.xlabel(r'RG Step')
    plt.plot(y5, y2)
    plt.show()
else:
    print ("Not fixed point.")
```

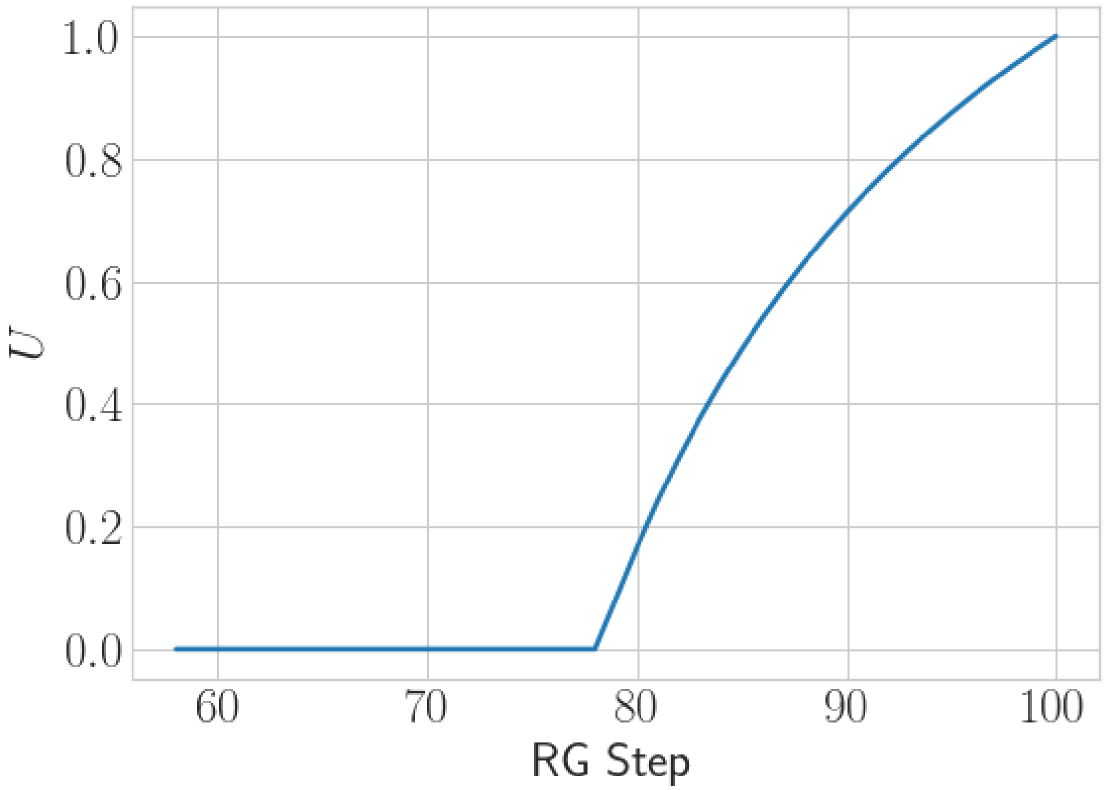


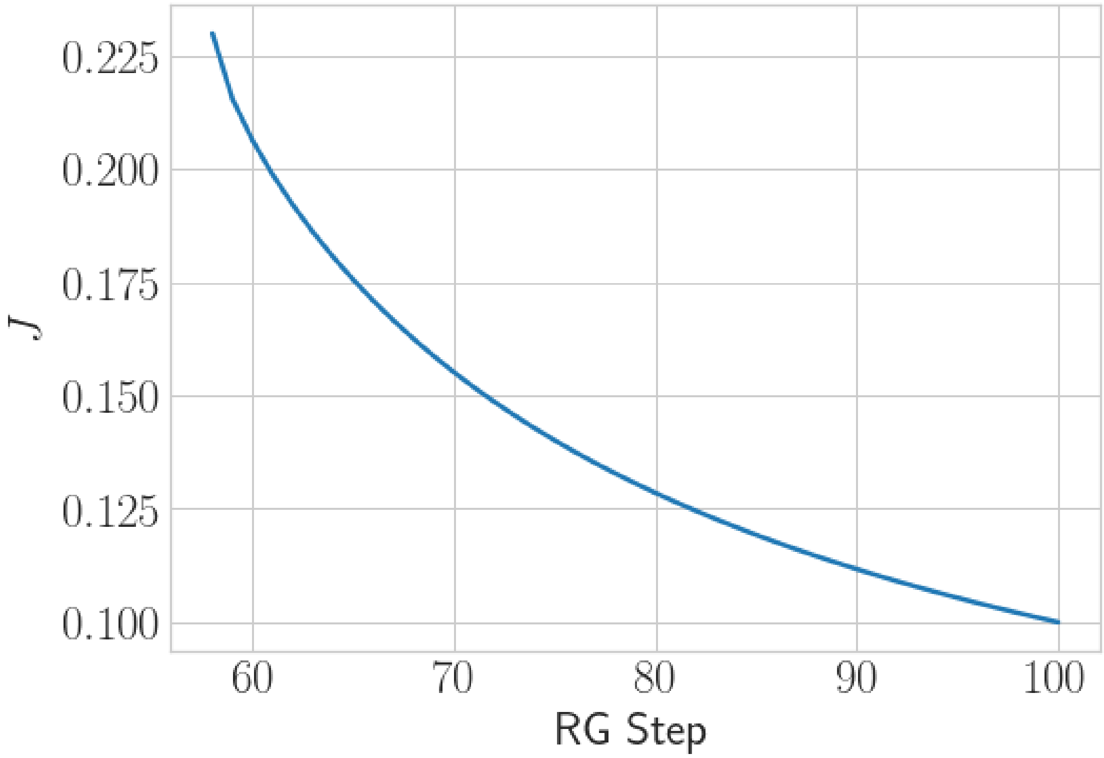


2.2.2 i. $J < K$

[9]:

```
w = 0.01
D0 = 20
U0 = 1
J0 = 0.1
K0 = 0.2
V0 = 0
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
if flag == True:
    plt.ylabel(r'$U$')
    plt.xlabel(r'RG Step')
    plt.plot(y5, y1)
    plt.show()
    plt.ylabel(r'$J$')
    plt.xlabel(r'RG Step')
    plt.plot(y5, y2)
    plt.show()
else:
    print ("Not fixed point.")
```


U
1.0
0.8
0.6
0.4
0.2
0.0
60
70
80
90
100
RG Step


J
0.225
0.200
0.175
0.150
0.125
0.100
60
70
80
90
100
RG Step

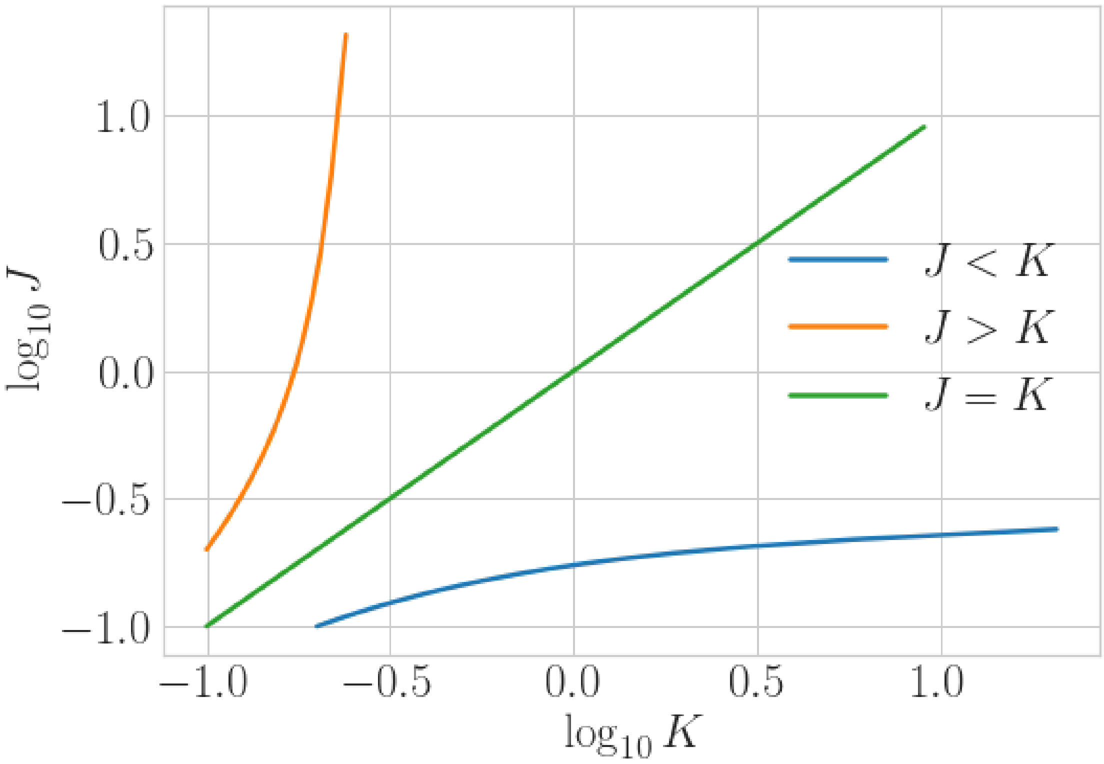
To wrap up the $V=0$ case, we look at an RG-invariant:

$\frac{\Delta J}{\Delta K} = \frac{J^2}{K^2} \implies \frac{1}{J} - \frac{1}{K} = \frac{1}{J_0} - \frac{1}{K_0}$

Note that this is an invariant even when V is turned on.

[10]:

```python
w = 0.1
D0 = 10
U0 = 1
J0 = 0.1
K0 = 0.2
V0 = 0
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
if flag == False:
    print ("It is not a fixed point.")
    exit
plt.xlabel(r"$\log_{10}K$")
plt.ylabel(r"$\log_{10}J$")
plt.plot(np.log10(y3), np.log10(y2), label=r'$J<K$')
J0 = 0.2
K0 = 0.1
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
if flag == False:
    print ("It is not a fixed point.")
    exit
plt.plot(np.log10(y3), np.log10(y2), label=r'$J>K$')
J0 = 0.1
K0 = 0.1
x, y1, y2, y3, y4, flag, y5 = complete_RG(w, D0, U0, V0, J0, K0)
if flag == False:
    print ("It is not a fixed point.")
    exit
plt.plot(np.log10(y3), np.log10(y2), label=r'$J=K$')
plt.legend()
plt.show()
```



2.3 Phase Diagram

3 2. $V > 0$

The inclusion of V will mean that there will not by any sharply defined phase of U^* any more. We will still be working in the regime where J, K flow to strong-coupling, and since those RG equations do not depend on V, their flows are unchanged. The behaviour of U will get complicated however. To make sense, we will see how the total (over a range of ω and bare U) number of fixed points where $U^* > U_0$ and the total number of fixed points where $U^* < U_0$, in each of the four quadrants of the phase diagram, varies against the bare value V_0.

3.1 a. Behaviour of distribution of fixed points as a function of bare V

We can classify the fixed points into three classes: $U* = 0$, $U^* > U_0$ and $U^* < U_0$. The number of fixed points in each class for $V = 0$ has already been clarified in the $V = 0$ section, specially in the phase diagram. For that, we will first create some helper functions. - count_fp(args): returns the fraction of fixed points with $U^* = 0$ (c_0), $U^* > U_0$ (c_1) and $U^* < U_0$ (c_2), for given values of D_0, V_0, J_0, K_0 - get_Vc(args): returns the critical V_0 at which $c_0 = c_1 + c_2$ - plot_count(args): just plots the fraction of fixed points in each class as a function of bare values, given the data - plot_frac(args): just plots the fraction of $U^* = 0$ or $U^* \neq 0$ fixed points a particular V_0, as a function of D

[11]:

```
def count_fp(D0, V0, J0, K0, sign, delta=0.01):
    w_range = np.arange(-D0/2, D0/2, delta)
    U_range = np.arange(sign*delta, sign*(5 + delta), sign*delta)
    data = itertools.product(w_range, [D0], U_range, [V0], [J0], [K0])
    count = np.zeros(3)
    for outp in Pool(processes=50).starmap(complete_RG, data):
        U0 = outp[1][0]
        U_fp = outp[1][-1]
        if outp[-2] == False and U_fp != 0:
            continue
        if U_fp == 0:
            count[0] += 1
        elif U_fp > U0:
            count[1] += 1
        elif U_fp < U0:
            count[2] += 1
    return count
```

[12]:

```
def get_Vc(V0_range, c0, c1):
    diff = (c1-c0)[0]
    for i in range(1,len(c1-c0)):
        if diff * (c1-c0)[i] <=0 :
            return V0_range[i]
    return -1
```

[13]:

```
def plot_count(V0_range, count, title):
    y = [np.array(c)/sum(count) for c in count]
    plt.plot(V0_range, y[0], color='r', label=r"$U^*=0$" )
    plt.plot(V0_range, y[1], color='b', label=r"$U^* > U_0$")
    plt.plot(V0_range, y[2], color='g', label=r"$U^* < U_0$")
    plt.legend()
    plt.title(title)
    plt.xlabel(r"$V_0$")
    plt.ylabel(r"fraction of fixed points")
    plt.show()
```

[14]:

```
def plot_frac(D0_range, V0, frac, title):
    plt.plot(D0_range, frac[0])
    plt.xlabel(r"$D_0$")
    plt.ylabel(r"ratio of $U^*=0$ and $U^*\neq 0$")
    plt.title(title+" $V_0 = {}$".format(V0[0]))
    plt.show()
    plt.plot(D0_range, frac[1])
    plt.xlabel(r"$D_0$")
    plt.ylabel(r"ratio of $U^*=0$ and $U^*\neq 0$")
    plt.title(title+" $V_0 = {}$".format(V0[1]))
    plt.show()
```

We will first check how the c_i vary as functions of V, in each quadrant.

[15]:

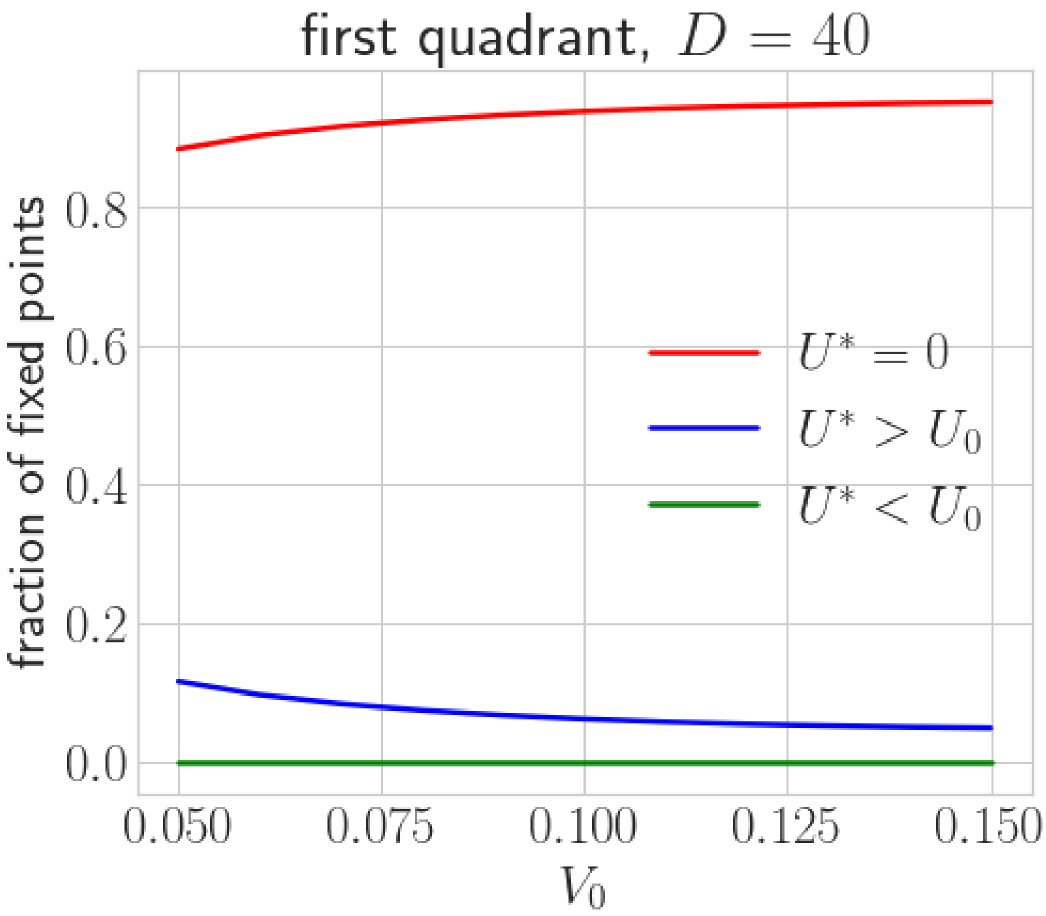
```
def sweep_V(J0, K0, sign, title, V0_range, D0_range=range(5, 21, 3)):
    for D0 in D0_range:
        c0, c1, c2 = [], [], []
        for V0 in tqdm(V0_range):
            count = count_fp(D0, V0, J0, K0, sign, delta=0.1)
            c0.append(count[0])
            c1.append(count[1])
            c2.append(count[2])
        c0, c1, c2 = np.array(c0), np.array(c1), np.array(c2)
        plot_count(V0_range, [c0, c1, c2], title+r", $D={}$".format(D0))
```

3.2 First Quadrant: $U > 0, J > K$

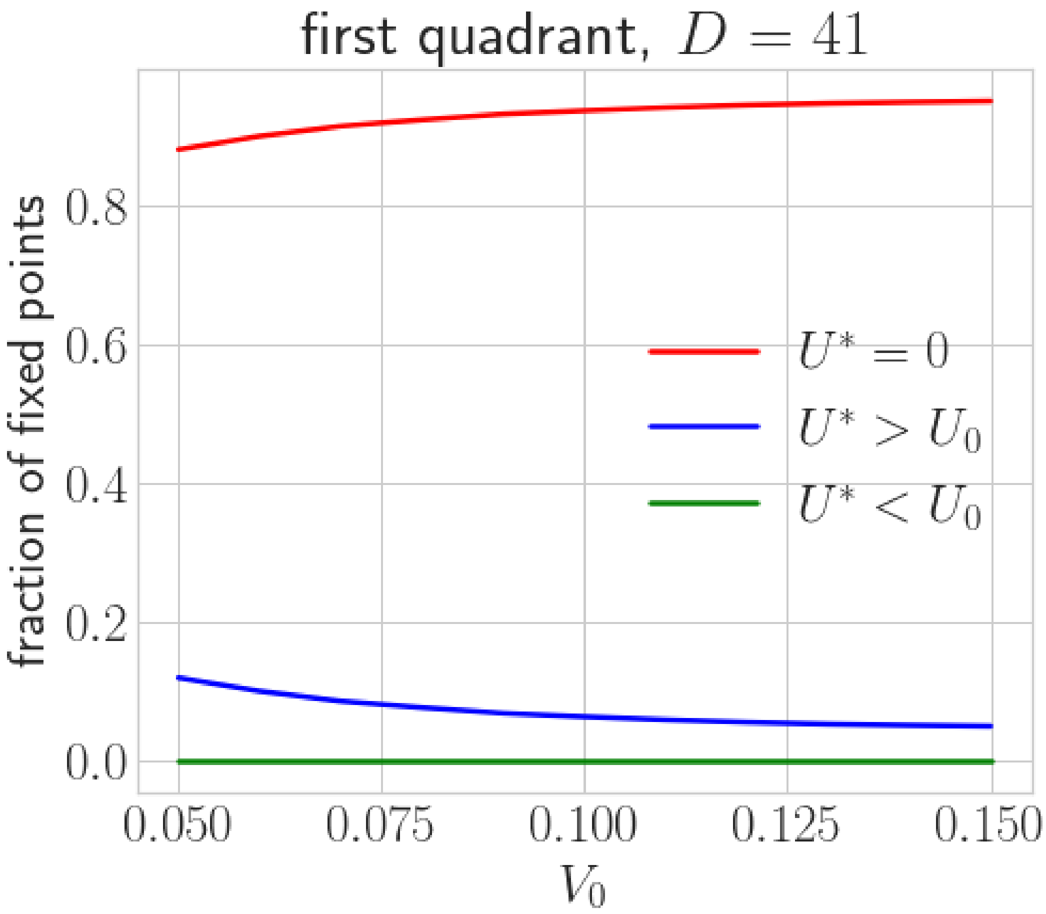
[23]:

```
#sweep_V(0.2, 0.1, 1, r"first quadrant", np.arange(0.049,0.051+0.0001,0.0001),␣
 ↪np.arange(3,4,2))
sweep_V(0.2, 0.1, 1, r"first quadrant", np.arange(0.05,0.15+0.01,0.01), np.
 ↪arange(40,42,1))
```

```
100%|      | 11/11 [01:32<00:00,  8.45s/it]
```



```
100%|          | 11/11 [01:24<00:00,  7.69s/it]
```

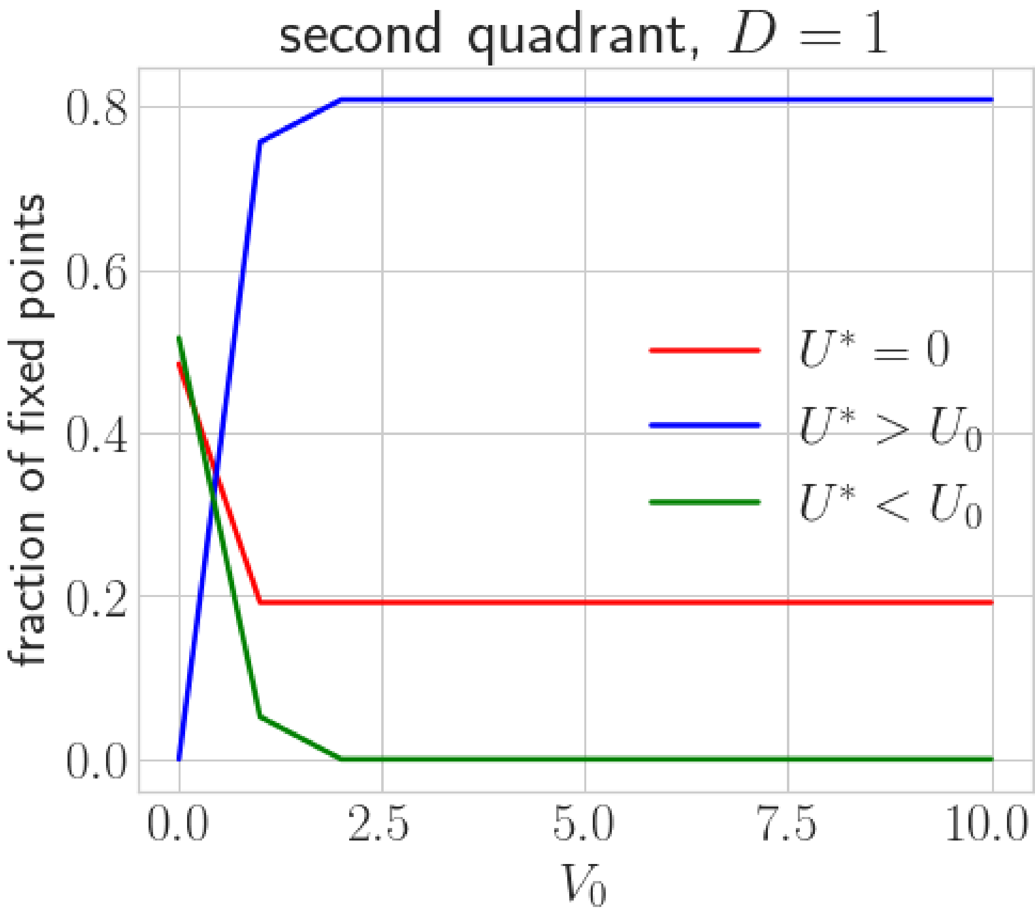


As D increases, dominant fixed point switches from $U^* > 0$ to $U^* = 0$ at some critical V_c. The critical V appears to decrease with D initially, but later increases (shown later). For large D, this critical V will be inaccessible, and the flip will be forbidden, leading to a phase where $U^* > U_0$.

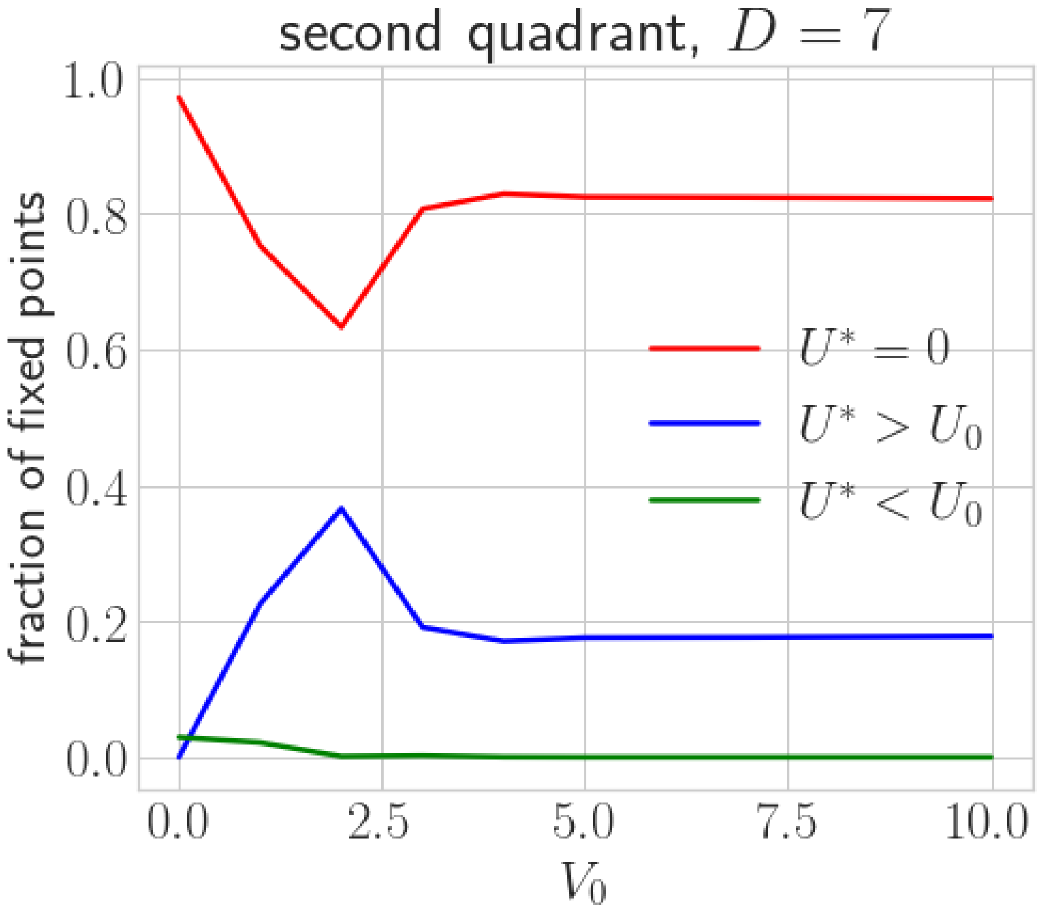
3.3 Second Quadrant: $U > 0, J < K$

[16]:
```
sweep_V(0.1, 0.2, 1, r"second quadrant", np.arange(0,10.1,1), np.arange(1,50,6))
```

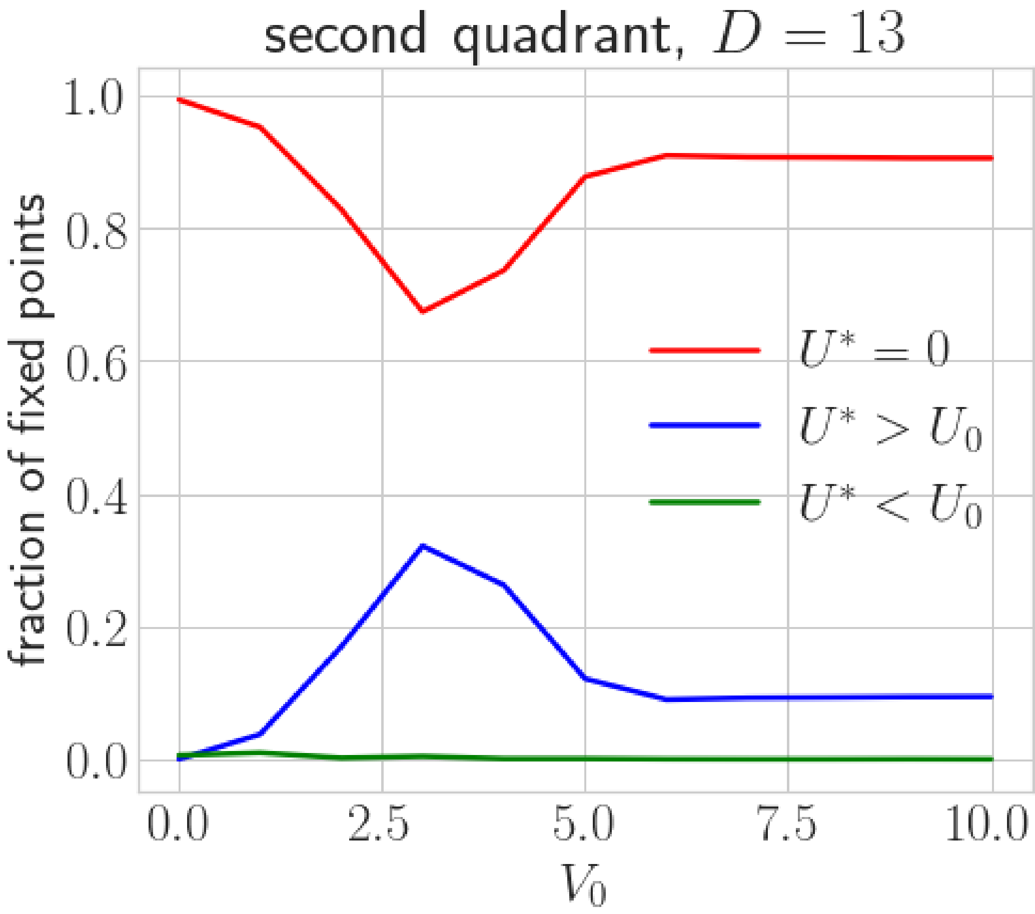
```
100%|          | 11/11 [00:03<00:00,  3.56it/s]
```



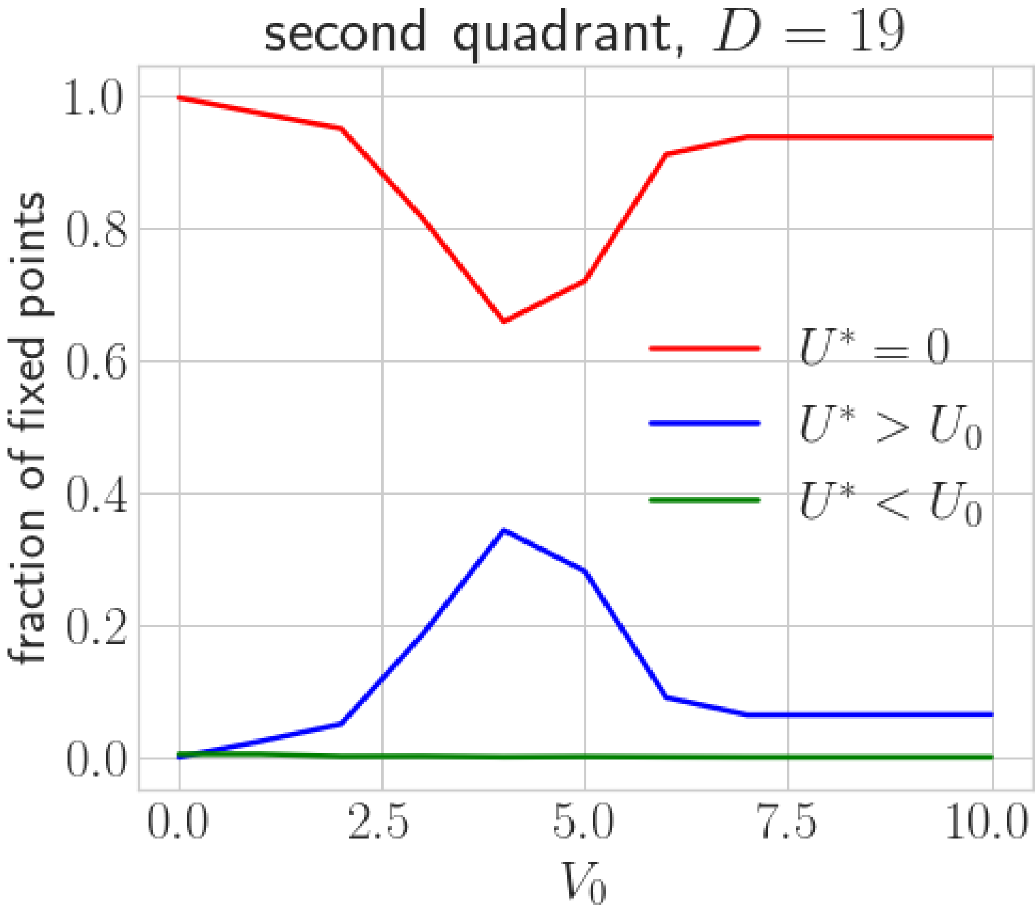
```
100%|      | 11/11 [00:06<00:00,  1.74it/s]
```



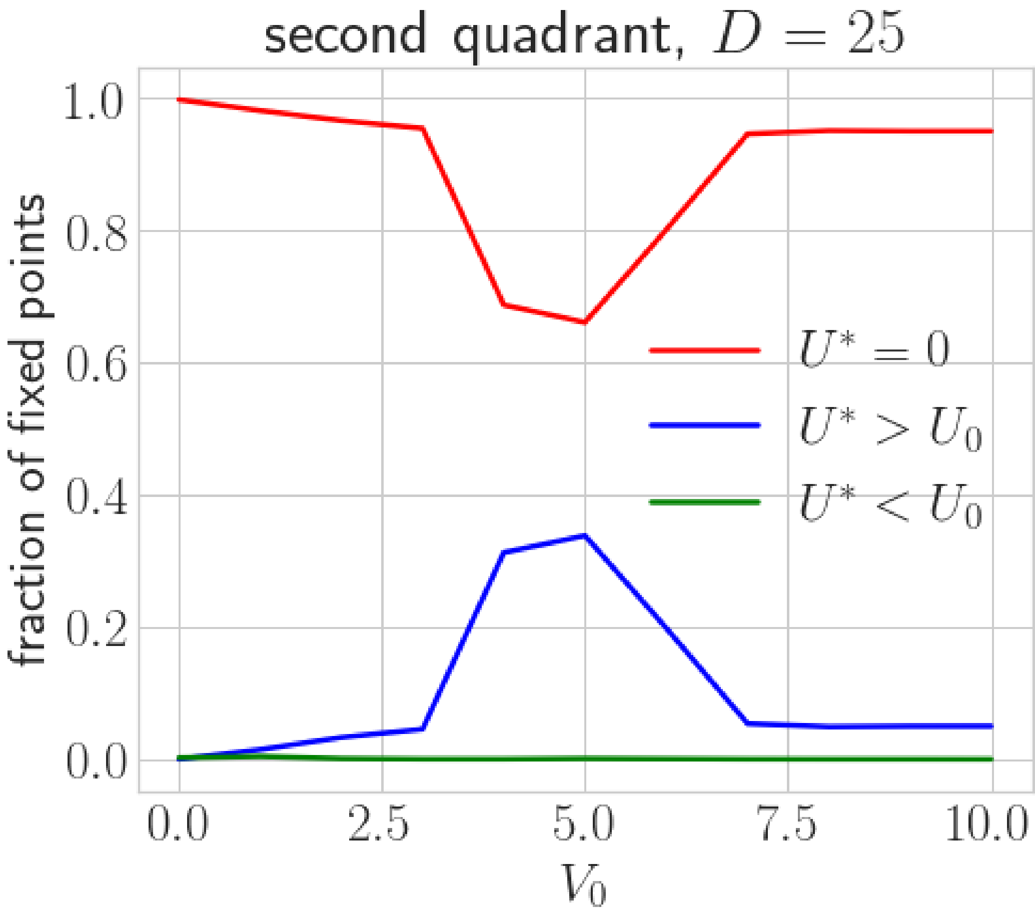
```
100%|      | 11/11 [00:12<00:00,  1.10s/it]
```



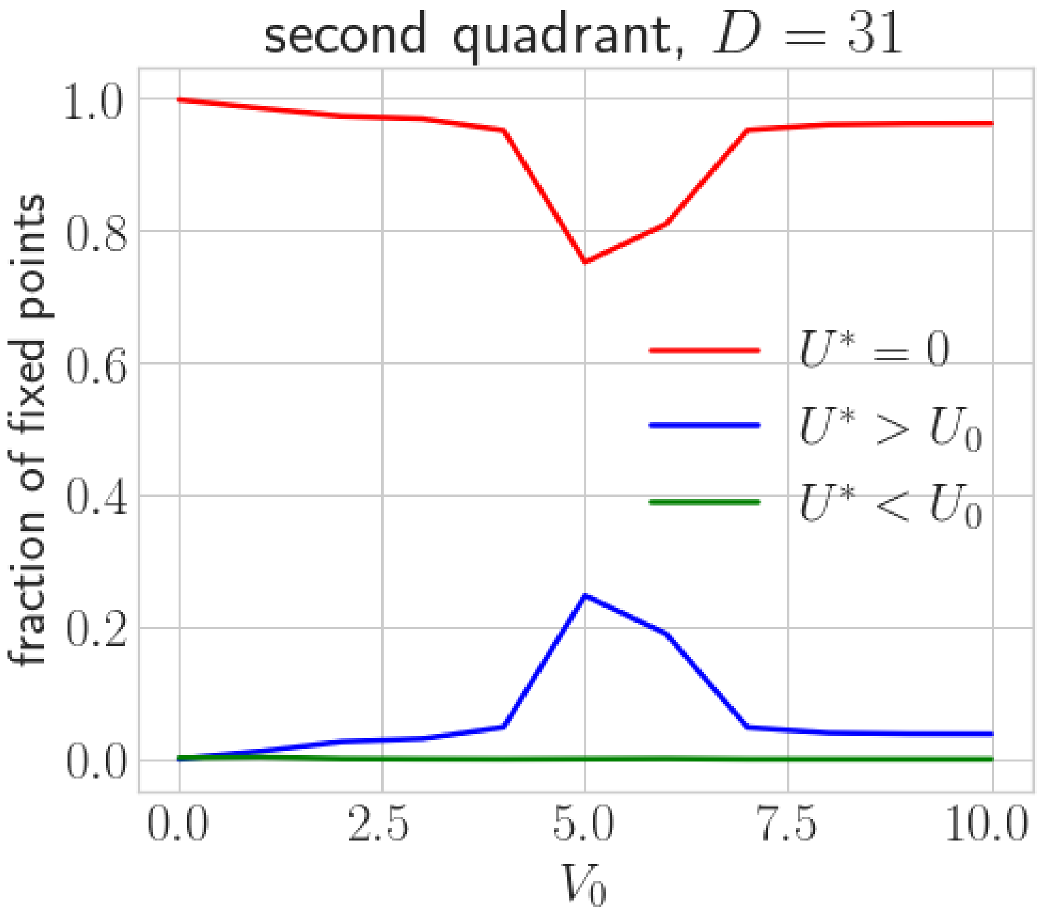
```
100%|      | 11/11 [00:19<00:00,  1.77s/it]
```



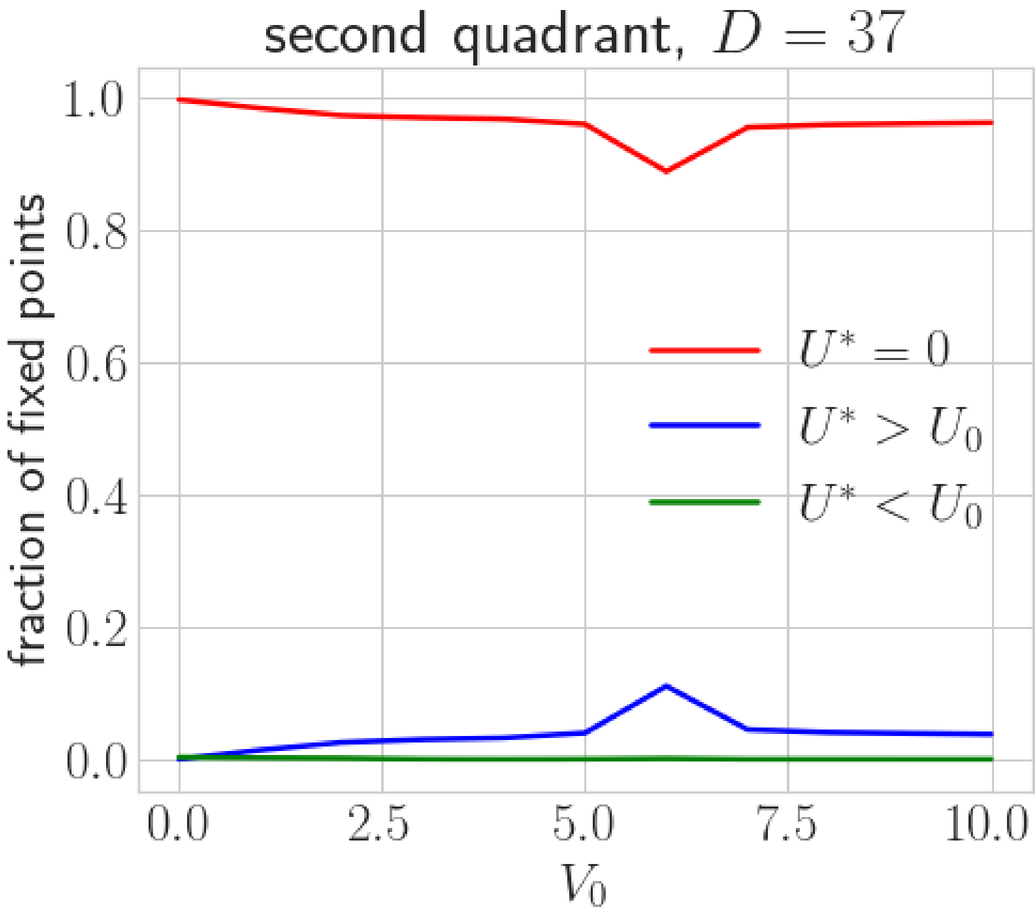
```
100%|          | 11/11 [00:28<00:00,  2.62s/it]
```



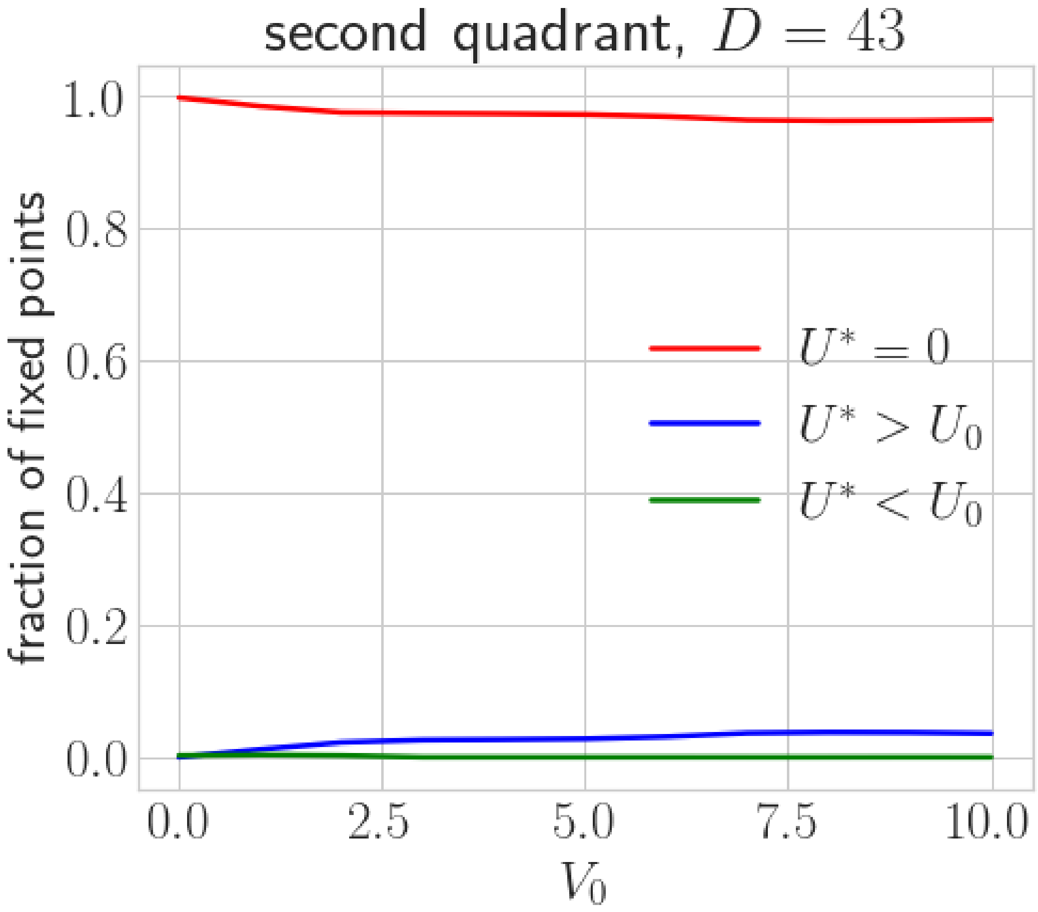
```
100%|          | 11/11 [00:38<00:00,  3.50s/it]
```



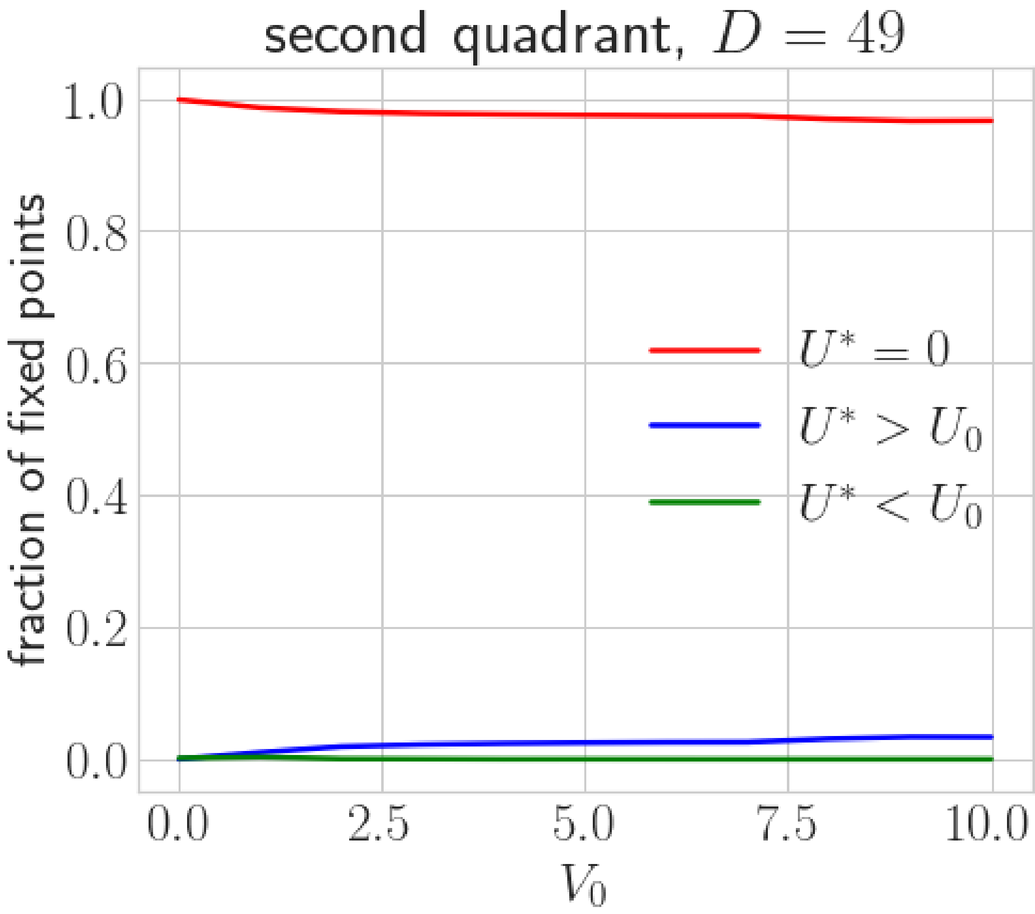
```
100%|      | 11/11 [00:47<00:00,  4.36s/it]
```



```
100%|      | 11/11 [00:58<00:00,  5.30s/it]
```

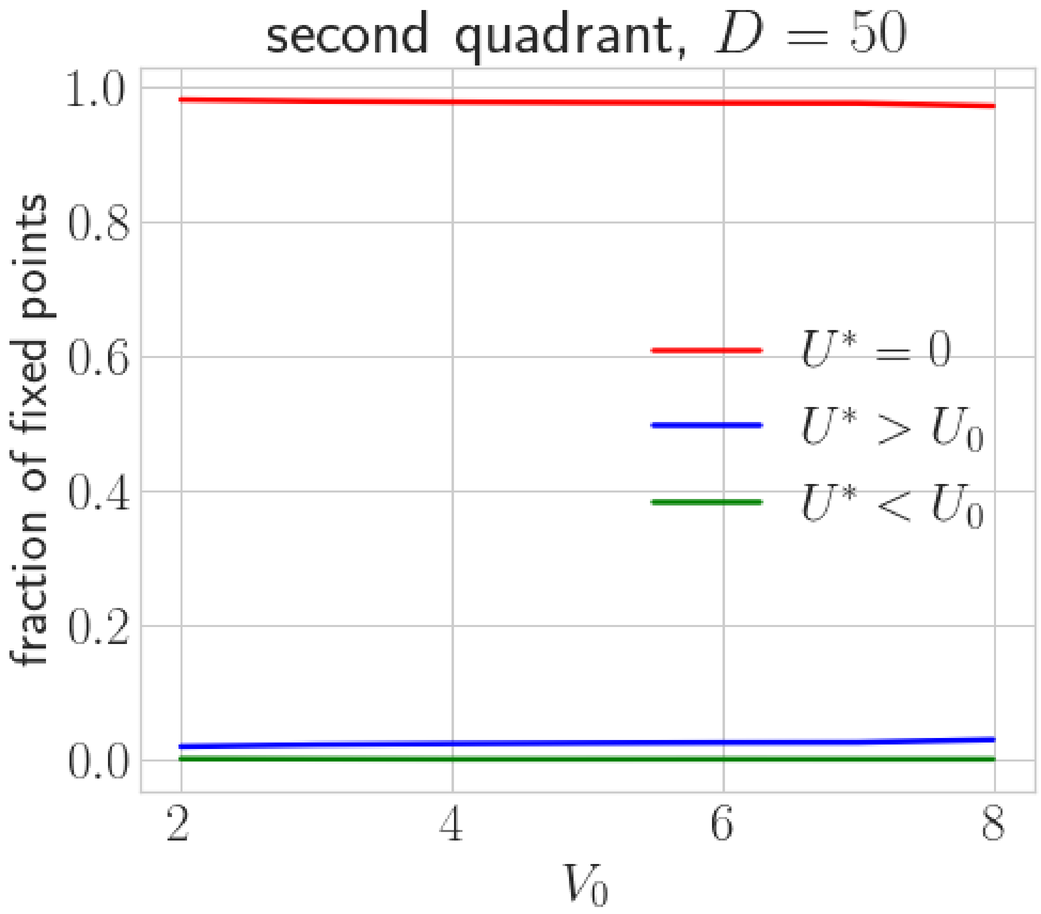


```
100%|          | 11/11 [01:06<00:00,  6.04s/it]
```

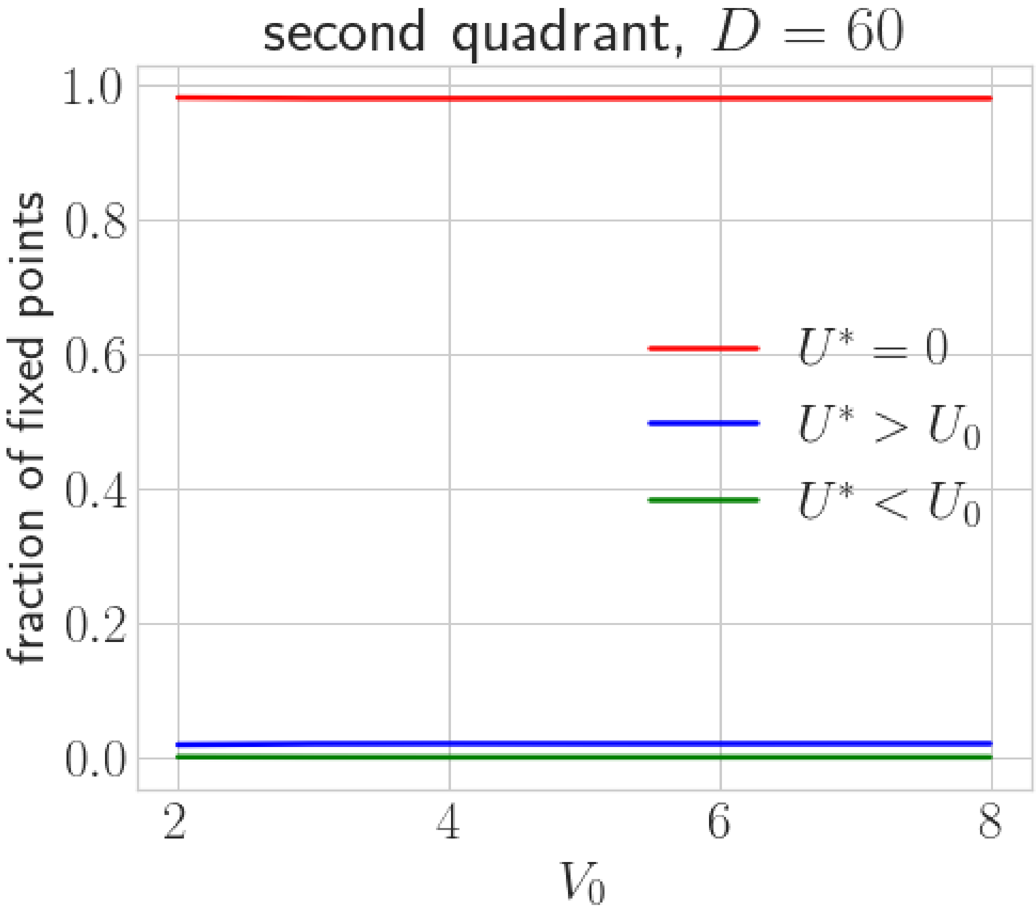


```
[17]: sweep_V(0.1, 0.2, 1, r"second quadrant", np.arange(2,9,1), np.arange(50,101,10))
```

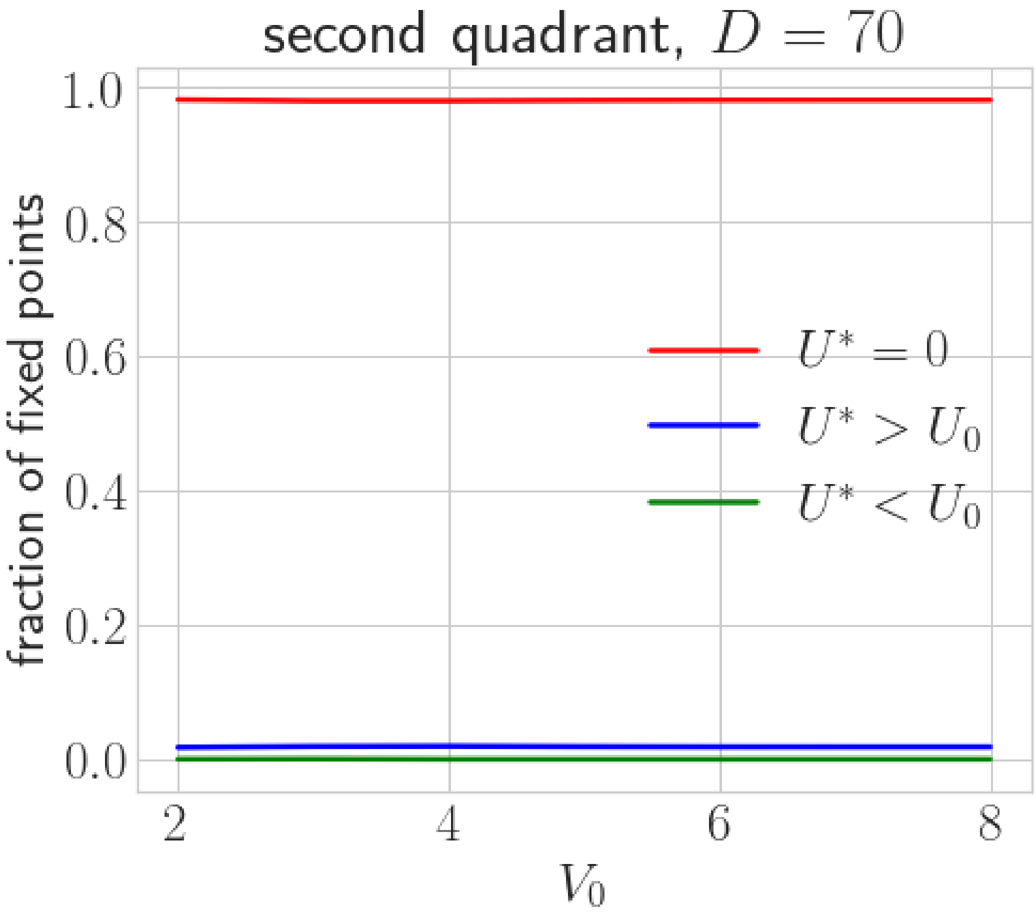
```
100%|          | 7/7 [00:42<00:00,  6.08s/it]
```



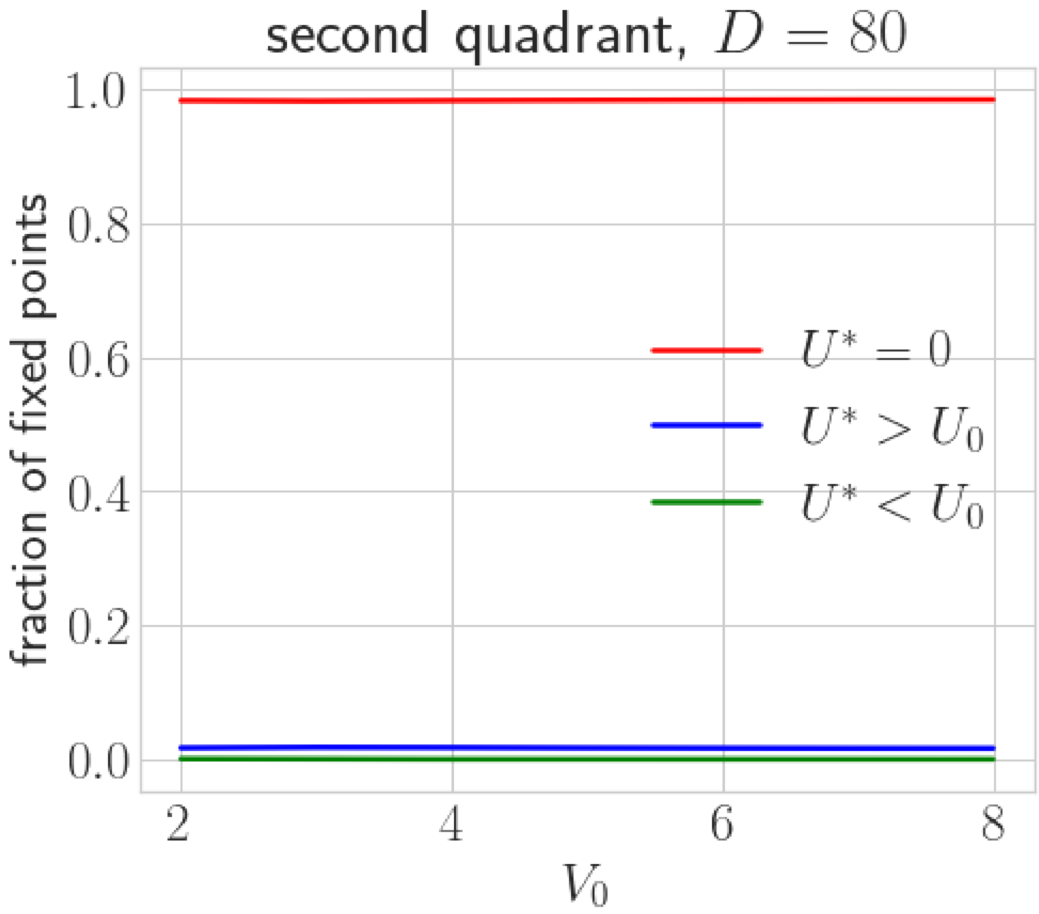
```
100%|        | 7/7 [00:54<00:00,  7.75s/it]
```



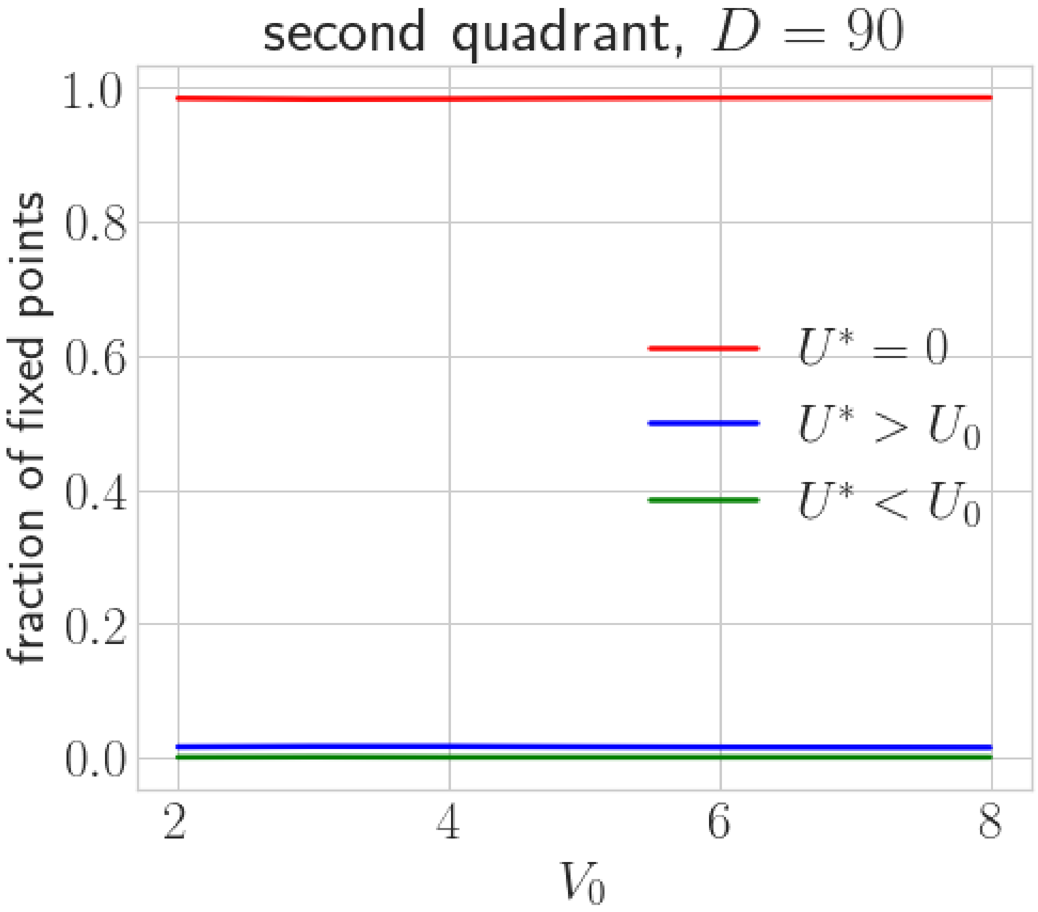
```
100%|      | 7/7 [01:03<00:00,  9.02s/it]
```



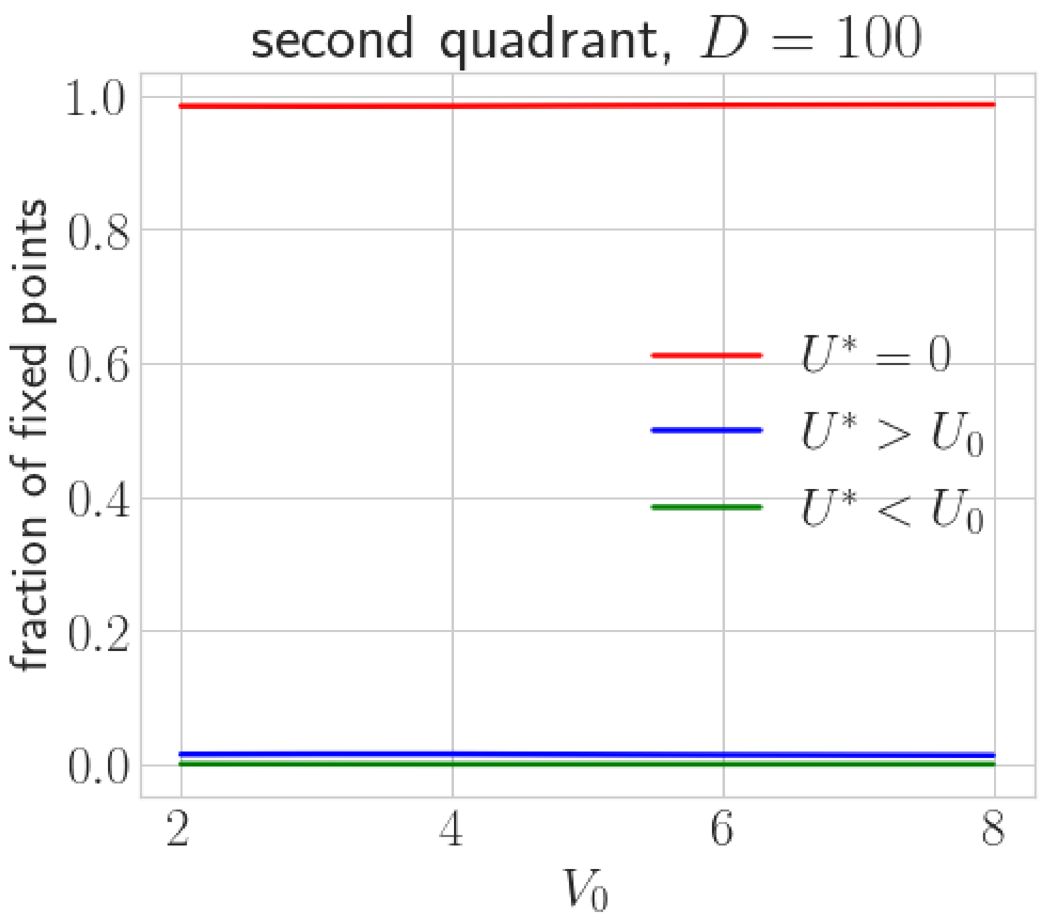
```
100%|          | 7/7 [01:13<00:00, 10.44s/it]
```



```
100%|          | 7/7 [01:24<00:00, 12.10s/it]
```



```
100%|      | 7/7 [01:35<00:00, 13.64s/it]
```

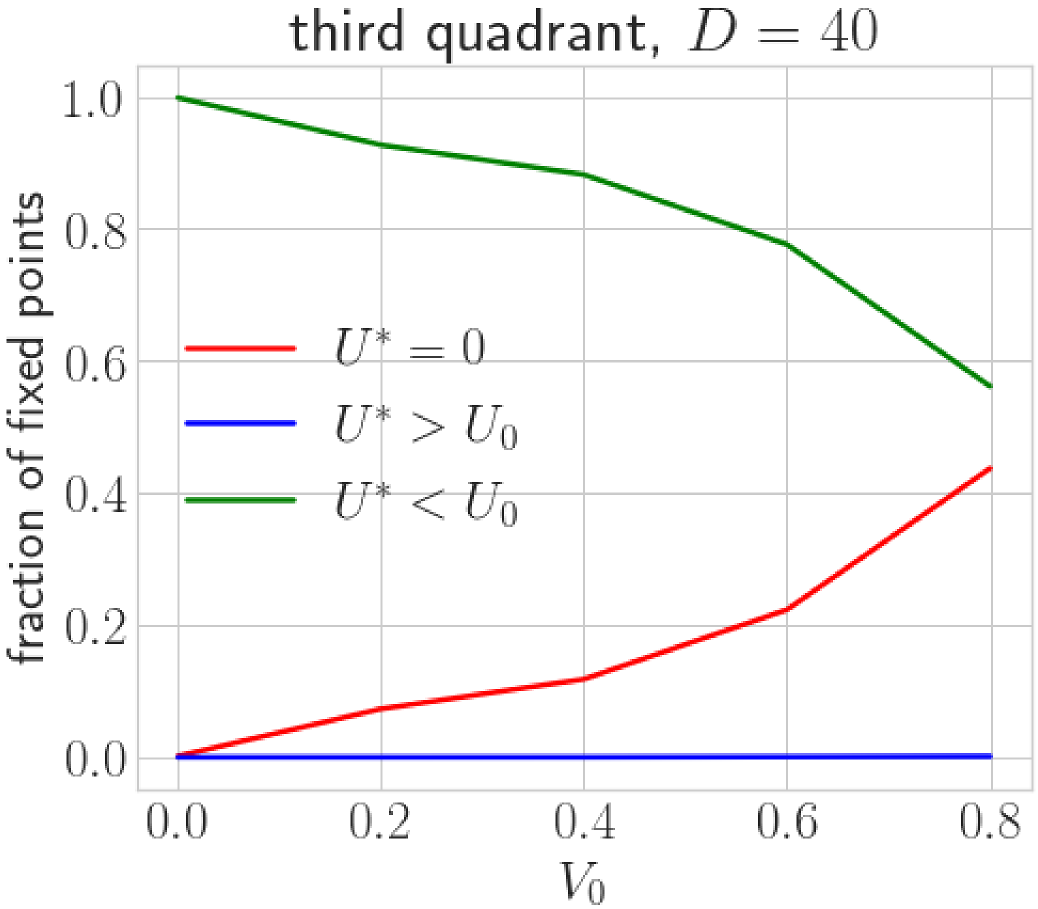


With increase in D, the hump keeps moving forward, and disappears at a sufficiently large D, so the dominant behaviour is unchanged (still $U^* = 0$).

3.4 Third Quadrant: $U < 0, J < K$

```
[26]: sweep_V(0.1, 0.2, -1, r"third quadrant", np.arange(0,1,0.2), np.arange(40,41,1))
```

```
100%|      | 5/5 [01:48<00:00, 21.66s/it]
```

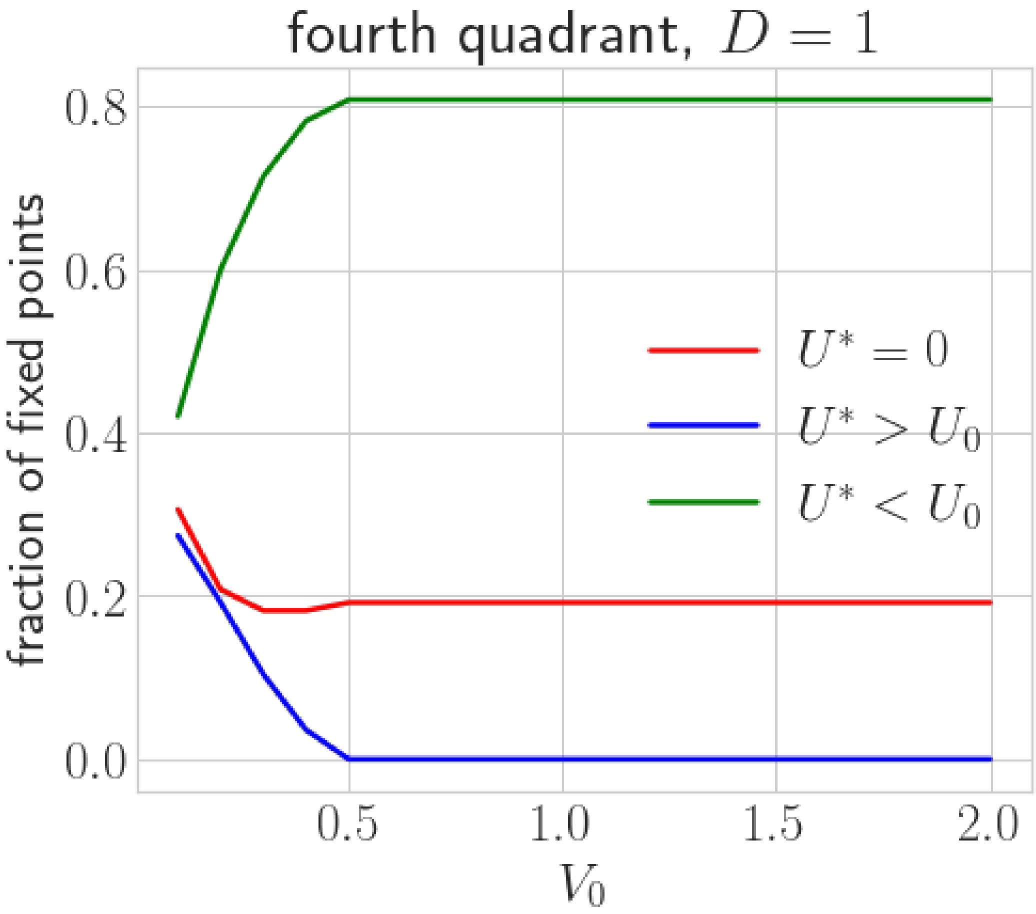


We see the opposite of the first quadrant behaviour here. There is again a flip at some critical V, critical V initially increases and then decreases. The fixed point phase should be $U^* < U_0$.

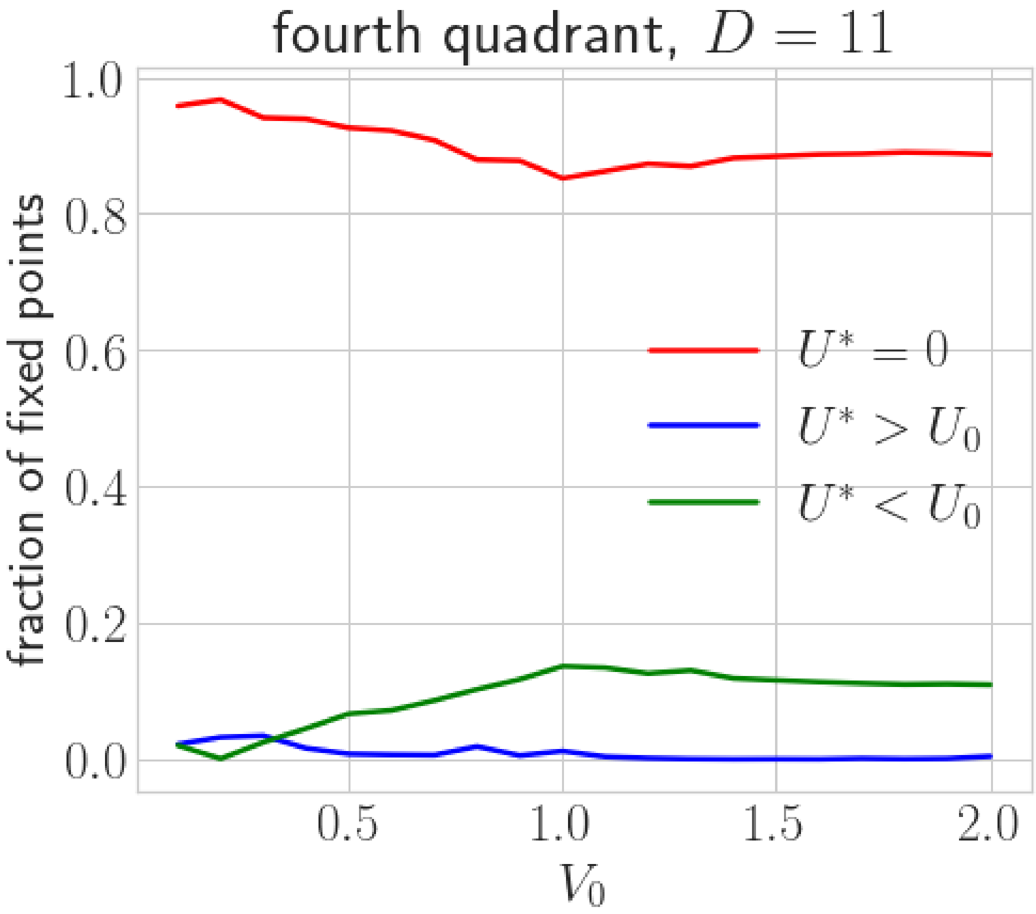
3.5 Fourth Quadrant: $U < 0, J > K$

```
[17]: sweep_V(0.2, 0.1, -1, r"fourth quadrant", np.arange(0.1,0.2,0.01), np.
       ↪arange(40,41,10))
```

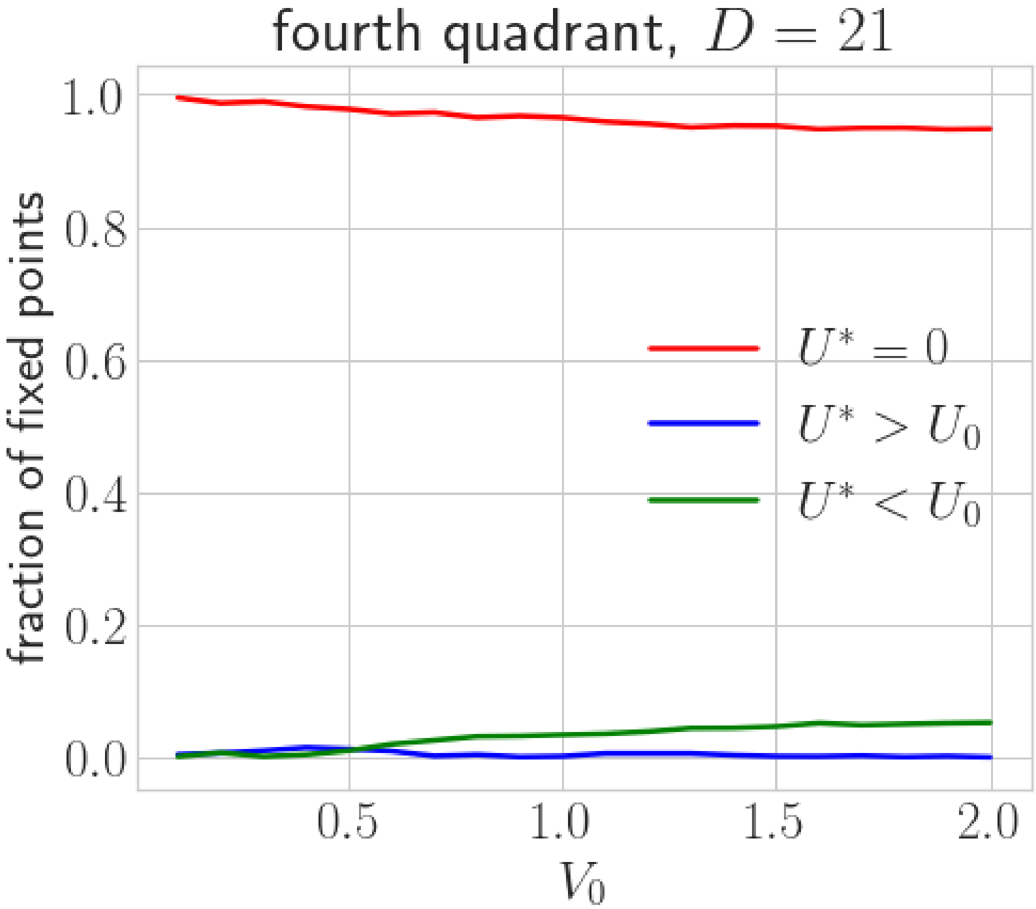
```
100%|          | 20/20 [00:06<00:00,  3.22it/s]
```



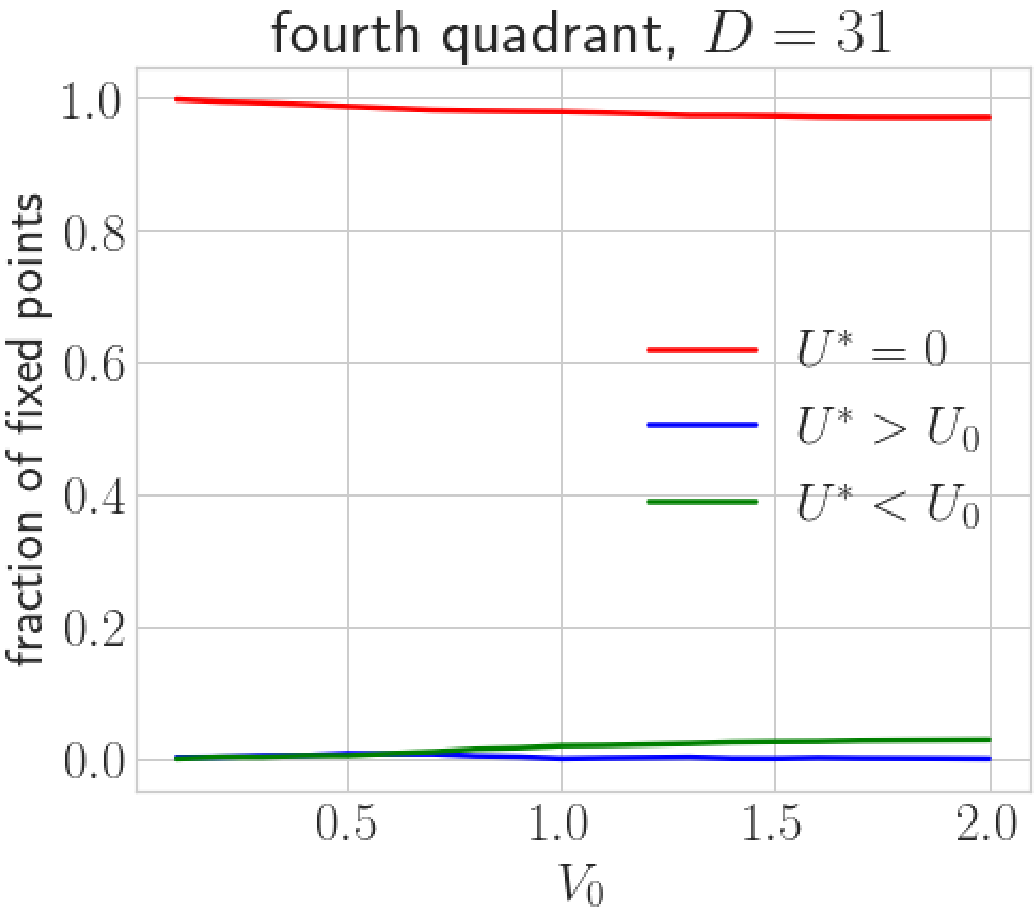
```
100%|          | 20/20 [00:19<00:00,  1.03it/s]
```



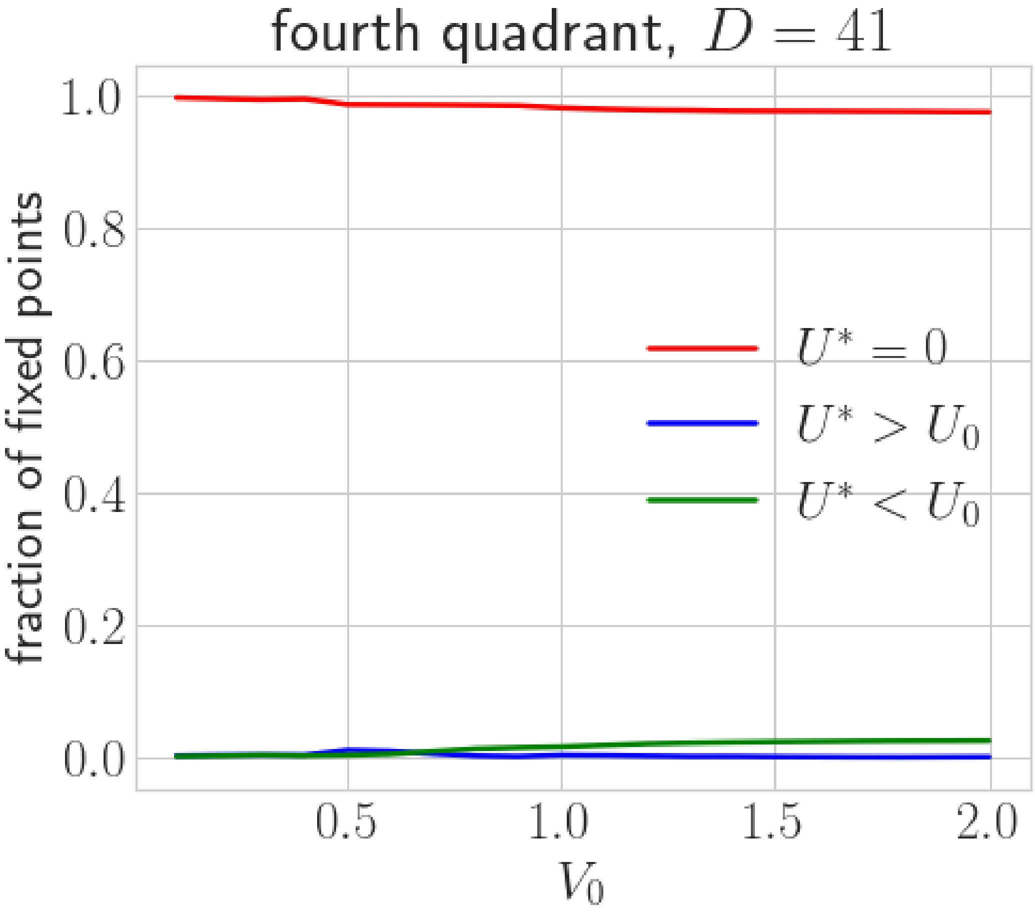
```
100%|      | 20/20 [00:42<00:00,  2.13s/it]
```



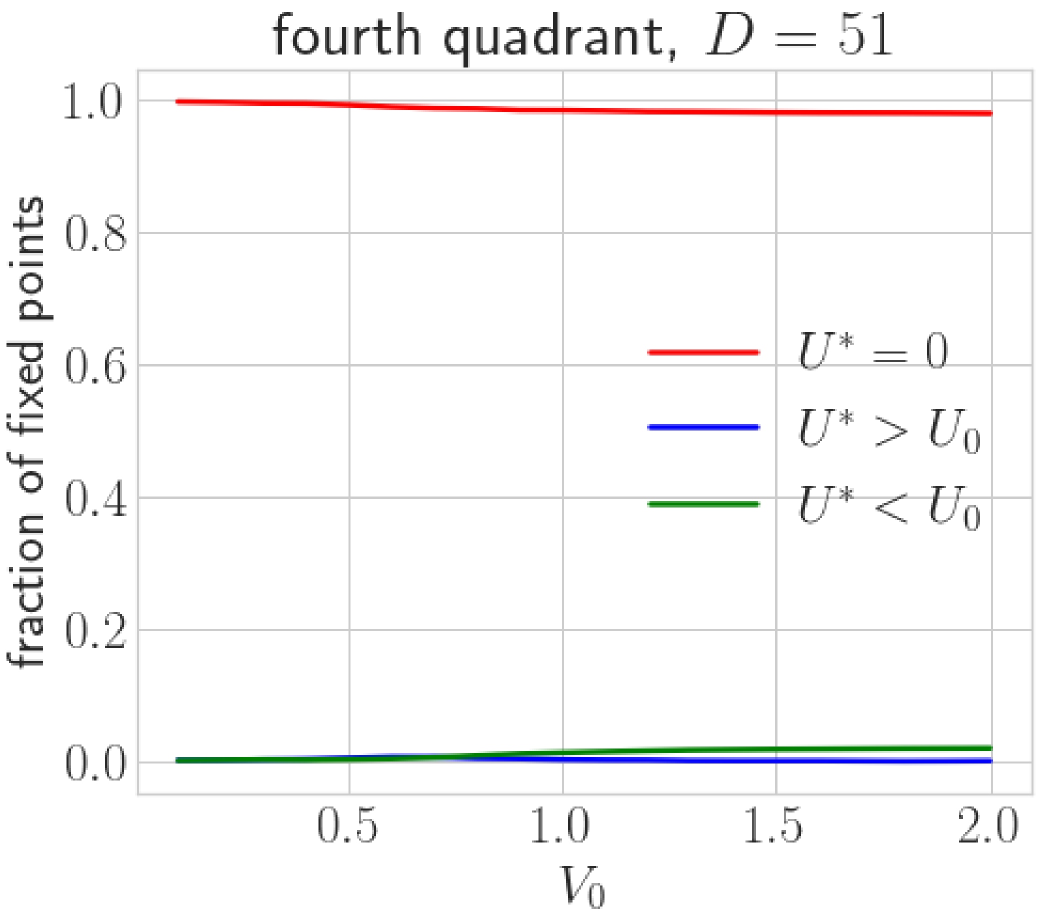
```
100%|      | 20/20 [01:10<00:00,  3.55s/it]
```



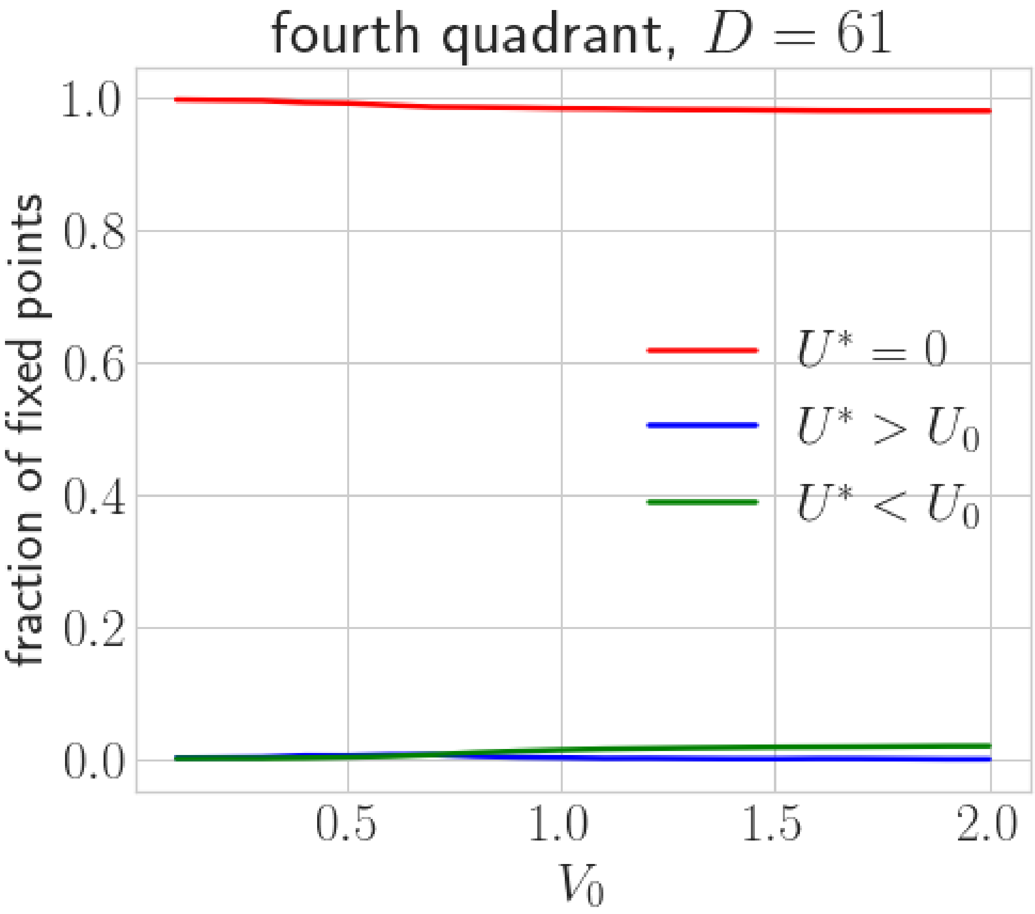
```
100%|      | 20/20 [01:42<00:00,  5.14s/it]
```



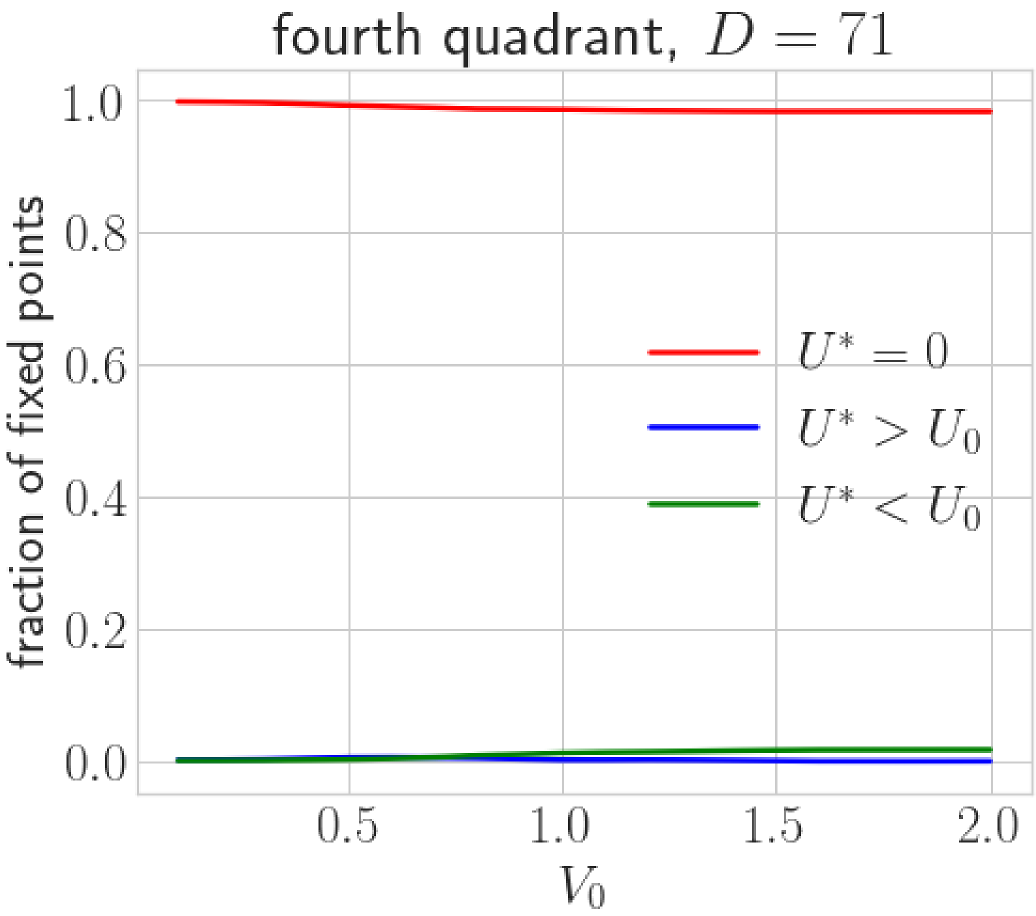
```
100%|      | 20/20 [02:11<00:00,  6.57s/it]
```



```
100%|     | 20/20 [02:41<00:00,  8.09s/it]
```



```
100%|     | 20/20 [03:11<00:00,  9.56s/it]
```



```
 45%|        | 9/20 [01:39<02:01, 11.07s/it]Process ForkPoolWorker-20899:
Process ForkPoolWorker-20886:
 45%|        | 9/20 [01:49<02:13, 12.14s/it]Process ForkPoolWorker-20890:
Process ForkPoolWorker-20896:
Process ForkPoolWorker-20889:
Process ForkPoolWorker-20867:
Process ForkPoolWorker-20891:
Process ForkPoolWorker-20883:
Process ForkPoolWorker-20868:
Process ForkPoolWorker-20900:
Process ForkPoolWorker-20875:
Process ForkPoolWorker-20897:
Process ForkPoolWorker-20878:
Process ForkPoolWorker-20866:
Process ForkPoolWorker-20872:
Process ForkPoolWorker-20857:
Traceback (most recent call last):
Process ForkPoolWorker-20865:
```

```
Process ForkPoolWorker-20859:

Process ForkPoolWorker-20856:
Process ForkPoolWorker-20860:
Process ForkPoolWorker-20861:
Process ForkPoolWorker-20869:
Process ForkPoolWorker-20855:
Process ForkPoolWorker-20894:
Process ForkPoolWorker-20880:
Process ForkPoolWorker-20892:
Traceback (most recent call last):
Traceback (most recent call last):
Process ForkPoolWorker-20851:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Process ForkPoolWorker-20879:
Process ForkPoolWorker-20888:
Process ForkPoolWorker-20885:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Process ForkPoolWorker-20895:
Traceback (most recent call last):
Process ForkPoolWorker-20882:
Process ForkPoolWorker-20898:
Process ForkPoolWorker-20893:
Traceback (most recent call last):
Process ForkPoolWorker-20877:
Process ForkPoolWorker-20870:
Process ForkPoolWorker-20884:
Process ForkPoolWorker-20854:
Process ForkPoolWorker-20876:
Process ForkPoolWorker-20874:
Process ForkPoolWorker-20862:
Process ForkPoolWorker-20871:
Process ForkPoolWorker-20852:
Process ForkPoolWorker-20863:
Process ForkPoolWorker-20858:
Process ForkPoolWorker-20864:
Traceback (most recent call last):
Process ForkPoolWorker-20873:
Process ForkPoolWorker-20853:
Process ForkPoolWorker-20887:
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-17-1f6786032d40> in <module>
----> 1 sweep_V(0.2, 0.1, -1, r"fourth quadrant", np.arange(0.1,2.1,0.1), np.
 ↪arange(1,100,10))

<ipython-input-15-2d41692cb6c0> in sweep_V(J0, K0, sign, title, V0_range,␣
 ↪D0_range)
      3         c0, c1, c2 = [], [], []
      4         for V0 in tqdm(V0_range):
----> 5             count = count_fp(D0, V0, J0, K0, sign, delta=0.1)
      6             c0.append(count[0])
      7             c1.append(count[1])

<ipython-input-11-e9dd1ea5e540> in count_fp(D0, V0, J0, K0, sign, delta)
      4     data = itertools.product(w_range, [D0], U_range, [V0], [J0], [K0])
      5     count = np.zeros(3)
----> 6     for outp in Pool(processes=50).starmap(complete_RG, data):
      7         U0 = outp[1][0]
      8         U_fp = outp[1][-1]

~/miniconda3/lib/python3.9/multiprocessing/pool.py in starmap(self, func,␣
 ↪iterable, chunksize)
    370         `func` and (a, b) becomes func(a, b).
    371         '''
--> 372         return self._map_async(func, iterable, starmapstar, chunksize).
 ↪get()
    373
    374     def starmap_async(self, func, iterable, chunksize=None,␣
 ↪callback=None,

~/miniconda3/lib/python3.9/multiprocessing/pool.py in get(self, timeout)
    763
    764     def get(self, timeout=None):
--> 765         self.wait(timeout)
    766         if not self.ready():
    767             raise TimeoutError

~/miniconda3/lib/python3.9/multiprocessing/pool.py in wait(self, timeout)
    760
    761     def wait(self, timeout=None):
--> 762         self._event.wait(timeout)
    763
    764     def get(self, timeout=None):

~/miniconda3/lib/python3.9/threading.py in wait(self, timeout)
    572             signaled = self._flag
    573                 if not signaled:
--> 574                     signaled = self._cond.wait(timeout)
    575                 return signaled
    576 

~/miniconda3/lib/python3.9/threading.py in wait(self, timeout)
    310             try:    # restore state no matter what (e.g., KeyboardInterrupt
    311                 if timeout is None:
--> 312                     waiter.acquire()
    313                     gotit = True
    314                 else:

KeyboardInterrupt:
```

In the fourth quadrant, the V causes significant changes only at very small D.

3.6 $\frac{c_0}{c_1}$ at $V_0 = 0.02$ and V_c, both vs D, for the 1st quadrant

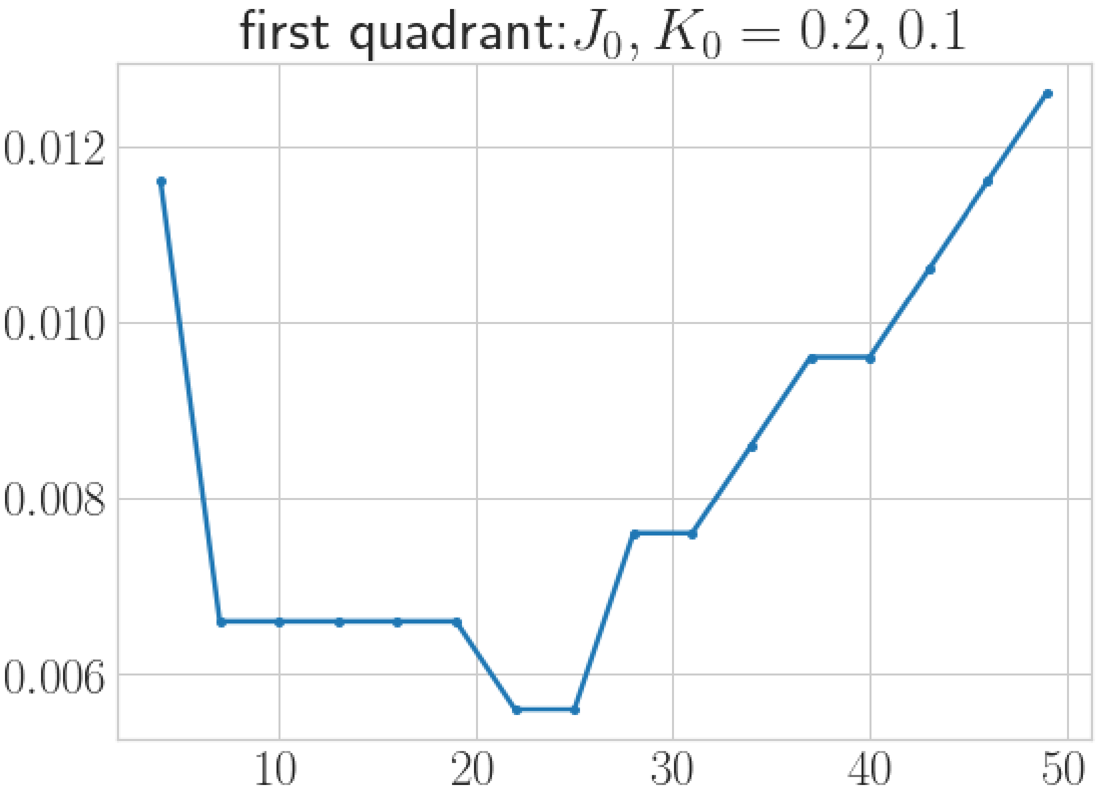
[26]:

```python
def plot_Vc(V0_start, D0_range, deltaV, J0, K0, sign, title):
    Vc = []
    frac = []
    D0 = D0_range[0]
    index = 1 if sign == 1 else 2
    while D0 in D0_range:
        print (D0)
        flag = False
        diff = 0
        V0_end = 2 if deltaV > 0 else 0
        for V0 in np.arange(V0_start, V0_end, deltaV):
            count = count_fp(D0, V0, J0, K0, sign, delta=0.05)
            if diff == 0:
                diff = count[0] - count[index]
            elif diff * (count[0] - count[index]) <= 0:
                V0_start = V0 - deltaV
                Vc.append([D0, V0])
                print (V0)
                flag = True
                D0 += D0_range[1] - D0_range[0]
                break
        if flag == False:
            deltaV *= -1


    Vc = np.array(Vc)
    frac = np.array(frac)
    plt.plot(Vc[:,0], Vc[:,1], marker=".")
    plt.title(title+r"$J_0, K_0 = {}, {}$".format(J0,K0))
    plt.show()
```

[27]:

```
#plot_Vc(0.0226, range(4,100,1), -0.00005, 0.2)
plot_Vc(0.0226, range(4,51,2), -0.0001, 0.2, 0.1, 1, "first quadrant:")
```

```
4
0.011599999999999989
7
0.006599999999999984
10
0.006599999999999984
13
0.006599999999999984
16
0.006599999999999984
19
0.006599999999999984
22
0.005599999999999984
25
0.005599999999999984
28
28
0.007599999999999984
31
0.007599999999999984
34
0.008599999999999984
37
0.009599999999999984
40
0.009599999999999984
43
0.010599999999999984
46
0.011599999999999985
49
0.012599999999999986
```



The critical V at which the transition from $U^* > U_0$ to $U^* = 0$ occurs is a function of D and J, K. It increases with increase in D, as well as increase in J.

[28]:
```
plot_Vc(0.52, range(3,40,2), 0.0001, 0.1, 0.2, -1, "third quadrant:")
```

```
3
0.522
6
Process ForkPoolWorker-135517:
Process ForkPoolWorker-135515:
Process ForkPoolWorker-135518:
Process ForkPoolWorker-135516:
Process ForkPoolWorker-135528:
Process ForkPoolWorker-135503:
Process ForkPoolWorker-135520:
Process ForkPoolWorker-135510:
Process ForkPoolWorker-135502:
Process ForkPoolWorker-135519:
Process ForkPoolWorker-135530:
Process ForkPoolWorker-135505:
Process ForkPoolWorker-135549:
Process ForkPoolWorker-135523:
```

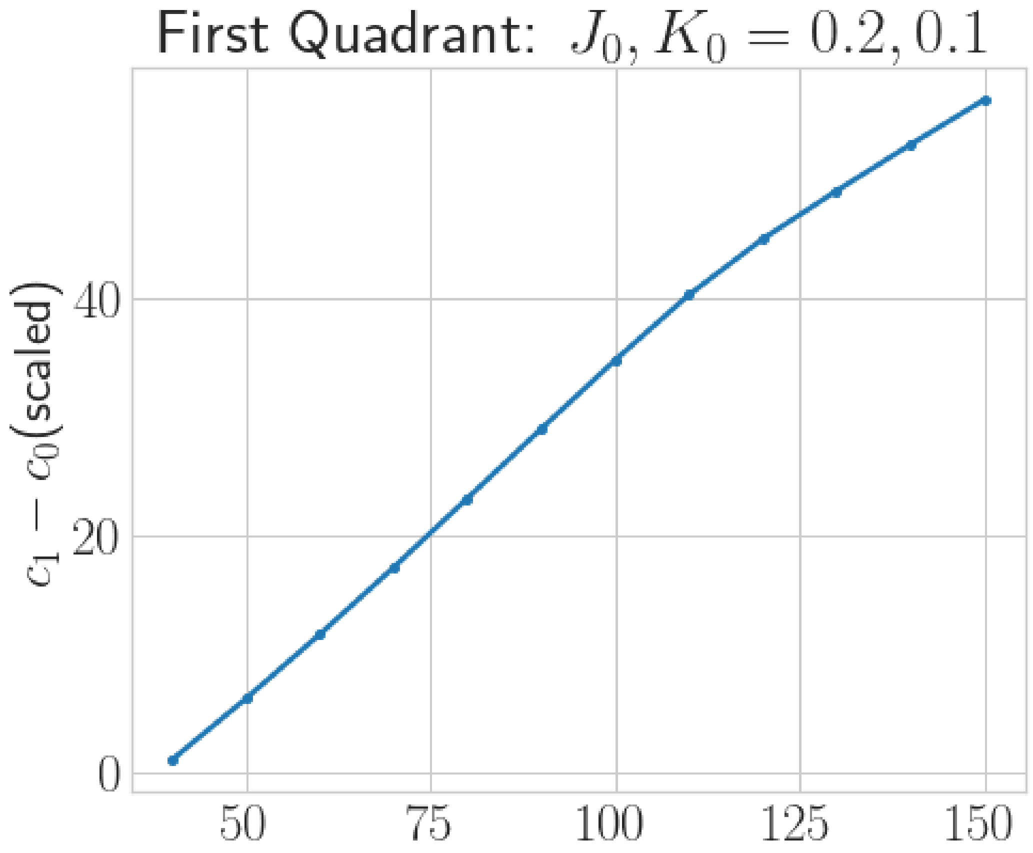
```
Process ForkPoolWorker-135506:
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-28-2740cf9c90e8> in <module>
----> 1 plot_Vc(0.52, range(3,40,3), 0.001, 0.1, 0.2, -1, "third quadrant:")

<ipython-input-26-d73a3bca45fd> in plot_Vc(V0_start, D0_range, deltaV, J0, K0,␣
 ↪sign, title)
     10         V0_end = 2 if deltaV > 0 else 0
     11         for V0 in np.arange(V0_start, V0_end, deltaV):
---> 12             count = count_fp(D0, V0, J0, K0, sign, delta=0.05)
     13             if diff == 0:
     14                 diff = count[0] - count[index]

<ipython-input-11-e9dd1ea5e540> in count_fp(D0, V0, J0, K0, sign, delta)
      4     data = itertools.product(w_range, [D0], U_range, [V0], [J0], [K0])
      5     count = np.zeros(3)
----> 6     for outp in Pool(processes=50).starmap(complete_RG, data):
      7         U0 = outp[1][0]
      8         U_fp = outp[1][-1]

~/miniconda3/lib/python3.9/multiprocessing/pool.py in starmap(self, func,␣
 ↪iterable, chunksize)
    370         `func` and (a, b) becomes func(a, b).
    371         '''
--> 372         return self._map_async(func, iterable, starmapstar, chunksize).
 ↪get()
    373
    374     def starmap_async(self, func, iterable, chunksize=None,␣
 ↪callback=None,

~/miniconda3/lib/python3.9/multiprocessing/pool.py in get(self, timeout)
    763
    764     def get(self, timeout=None):
--> 765         self.wait(timeout)
    766         if not self.ready():
    767             raise TimeoutError

~/miniconda3/lib/python3.9/multiprocessing/pool.py in wait(self, timeout)
    760
    761     def wait(self, timeout=None):
--> 762         self._event.wait(timeout)
    763
    764     def get(self, timeout=None):

~/miniconda3/lib/python3.9/threading.py in wait(self, timeout)
```

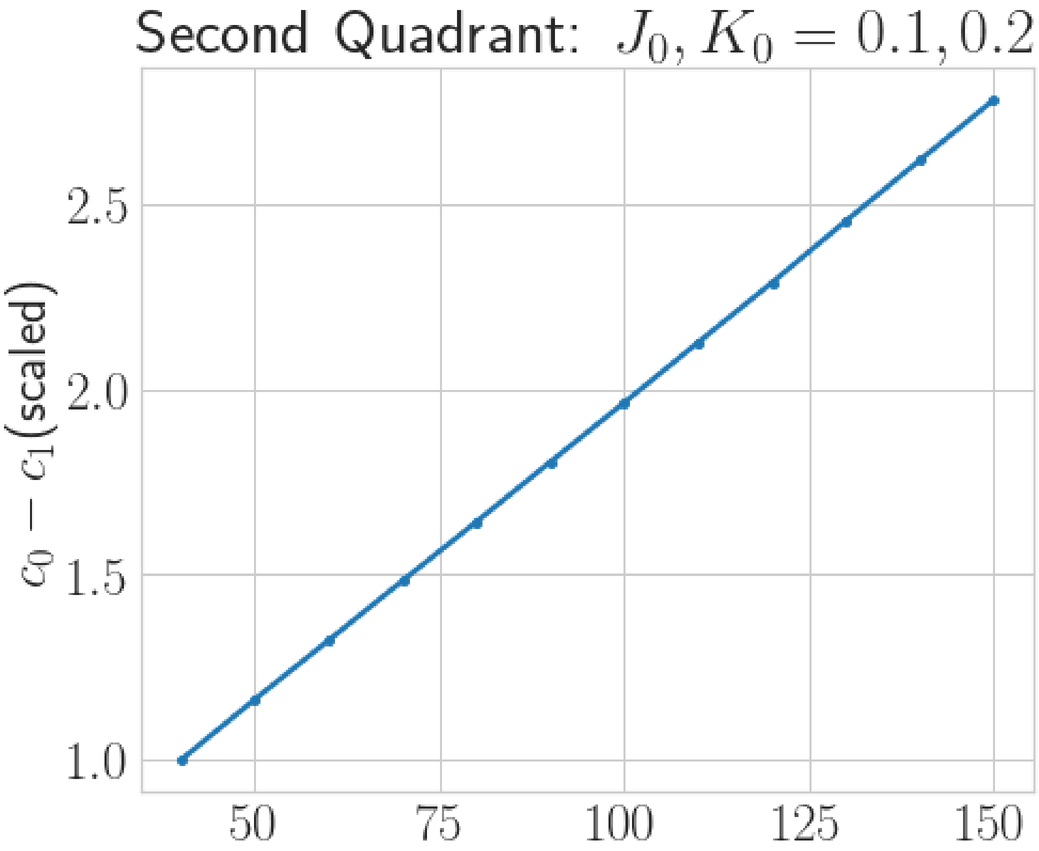
```
    572             signaled = self._flag
    573             if not signaled:
--> 574                 signaled = self._cond.wait(timeout)
    575             return signaled
    576 

~/miniconda3/lib/python3.9/threading.py in wait(self, timeout)
    310         try:    # restore state no matter what (e.g., KeyboardInterrupt
    311             if timeout is None:
--> 312                 waiter.acquire()
    313                 gotit = True
    314             else:

KeyboardInterrupt: 
```

[20]:
```python
def plot_frac(V0, D0_range, J0, K0, sign, title, inds):
    i, j = inds
    diff = []
    for D0 in D0_range:
        #print (D0)
        count = count_fp(D0, V0, J0, K0, sign, delta=0.1)
        diff.append(count[i]-count[j])

    diff = np.array(diff)
    diff = diff/min(diff)
    plt.plot(D0_range, diff, marker=".")
    plt.ylabel(r"$c_{} - c_{}$(scaled)".format(i,j))
    plt.title(title+r"$J_0, K_0 = {}, {}$".format(J0,K0))
    plt.show()
```

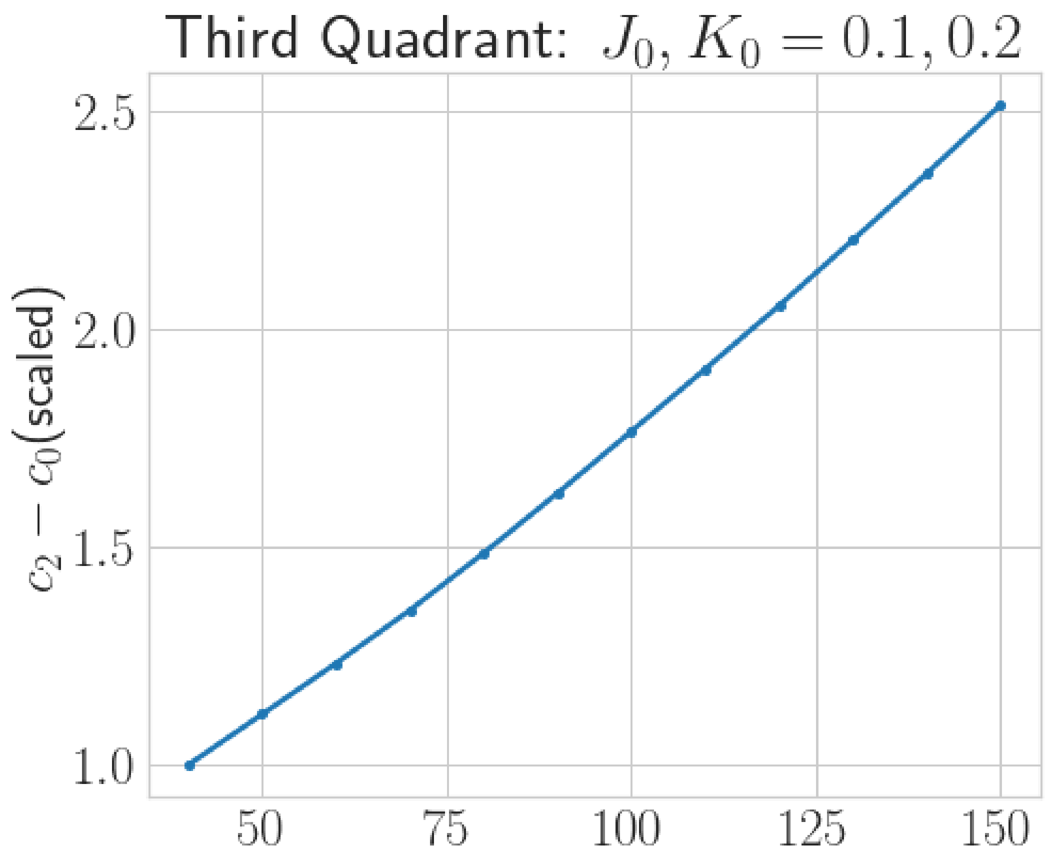
[21]:
```python
plot_frac(0.009, range(40,151,10), 0.2, 0.1, 1, r"First Quadrant: ", [1,0])
```



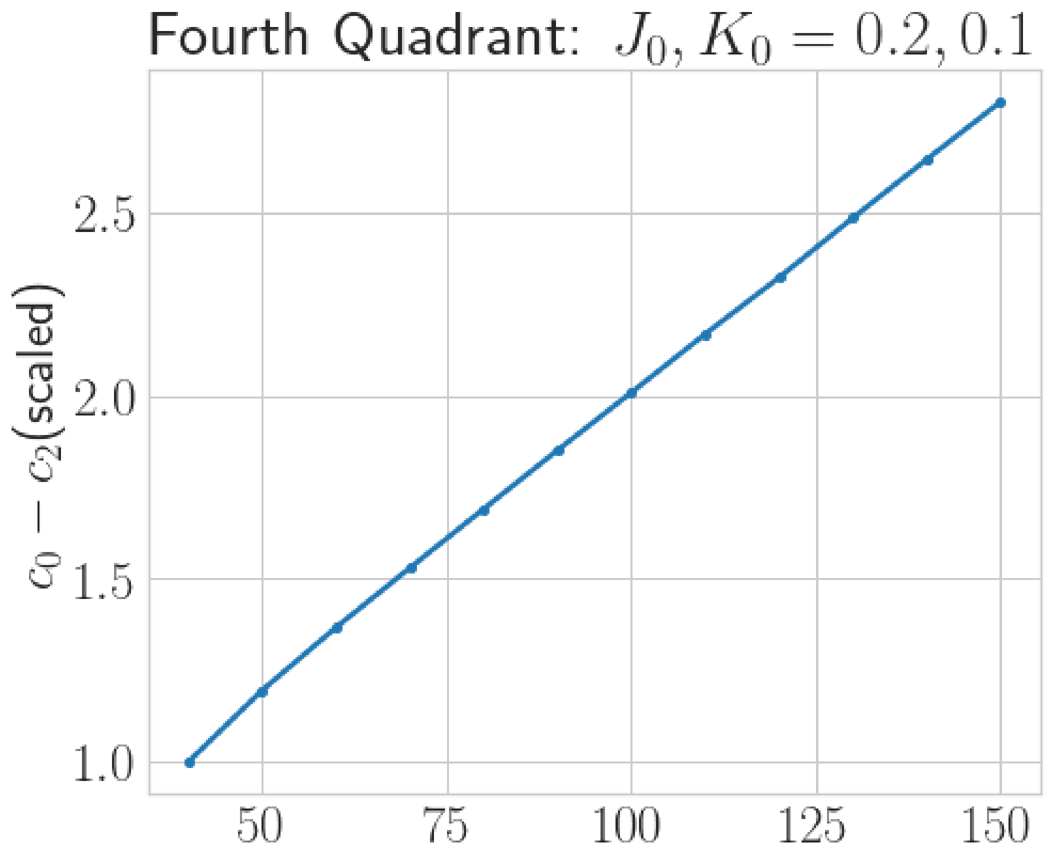
```
[22]: plot_frac(0.1, range(40,151,10), 0.1, 0.2, 1, r"Second Quadrant: ", [0,1])
```



```
[23]: plot_frac(0.1, range(40,151,10), 0.1, 0.2, -1, r"Third Quadrant: ", [2,0])
```



```
[24]: plot_frac(1.5, range(40,151,10), 0.2, 0.1, -1, r"Fourth Quadrant: ", [0,2])
```



3.7 Comparison of J^* and V^*

[19]:

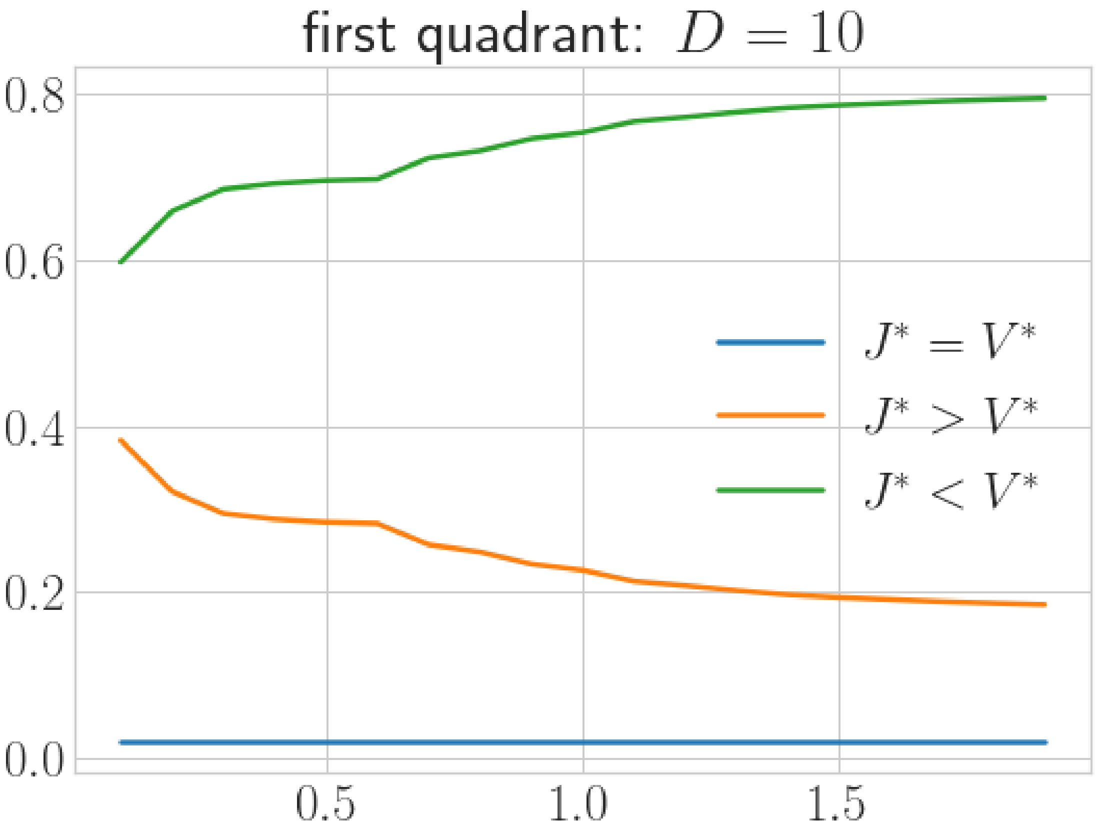
```
def count_Vfp(D0, V0, J0, K0, sign, delta=0.1):
    w_range = np.arange(-D0/2, D0/2, delta)
    U_range = np.arange(sign*delta, sign*(5 + delta), sign*delta)
    data = itertools.product(w_range, [D0], U_range, [V0], [J0], [K0])
    c0, cJ, cV = 0, 0, 0
    for outp in Pool(processes=50).starmap(complete_RG, data):
        x, y1, y2, y3, y4, flag, y5 = outp
        if flag == True or (y2[-1] == 0 and y4[-1] == 0):
            if y2[-1] > y4[-1]:
                cJ += 1
            elif y2[-1] < y4[-1]:
                cV += 1
            else:
                c0 += 1
    return c0, cJ, cV
```

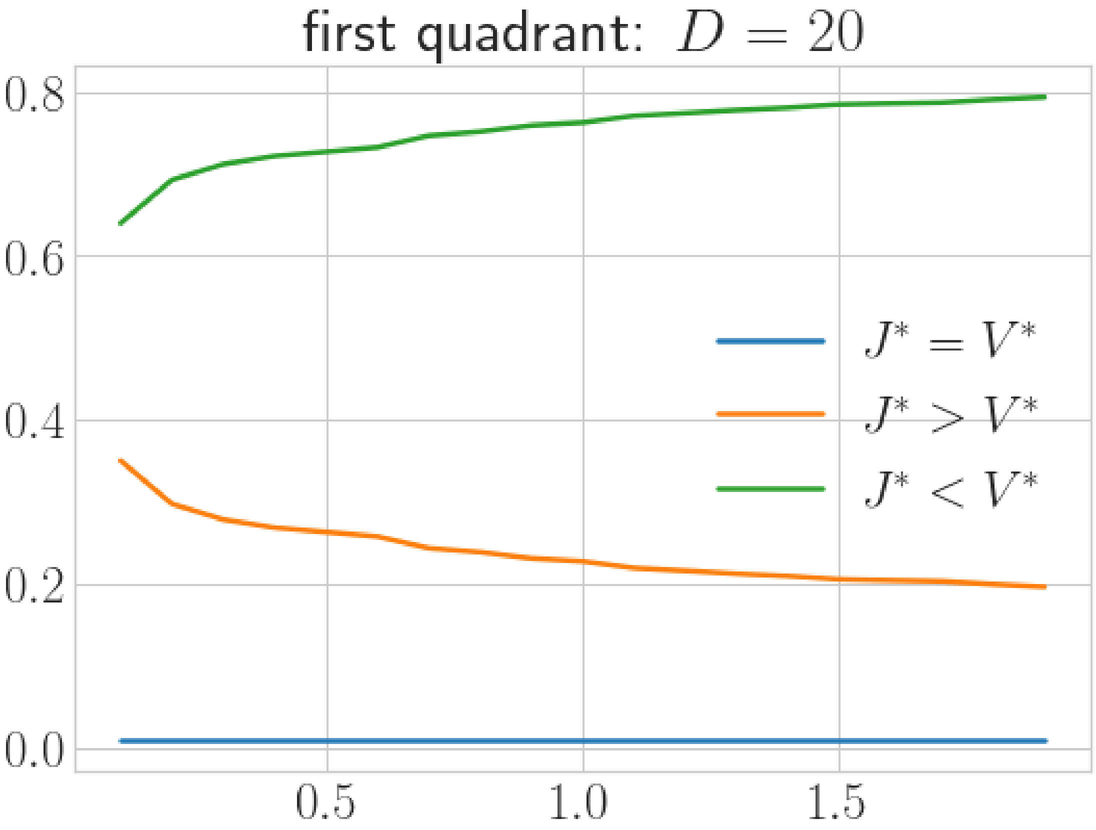
[20]:

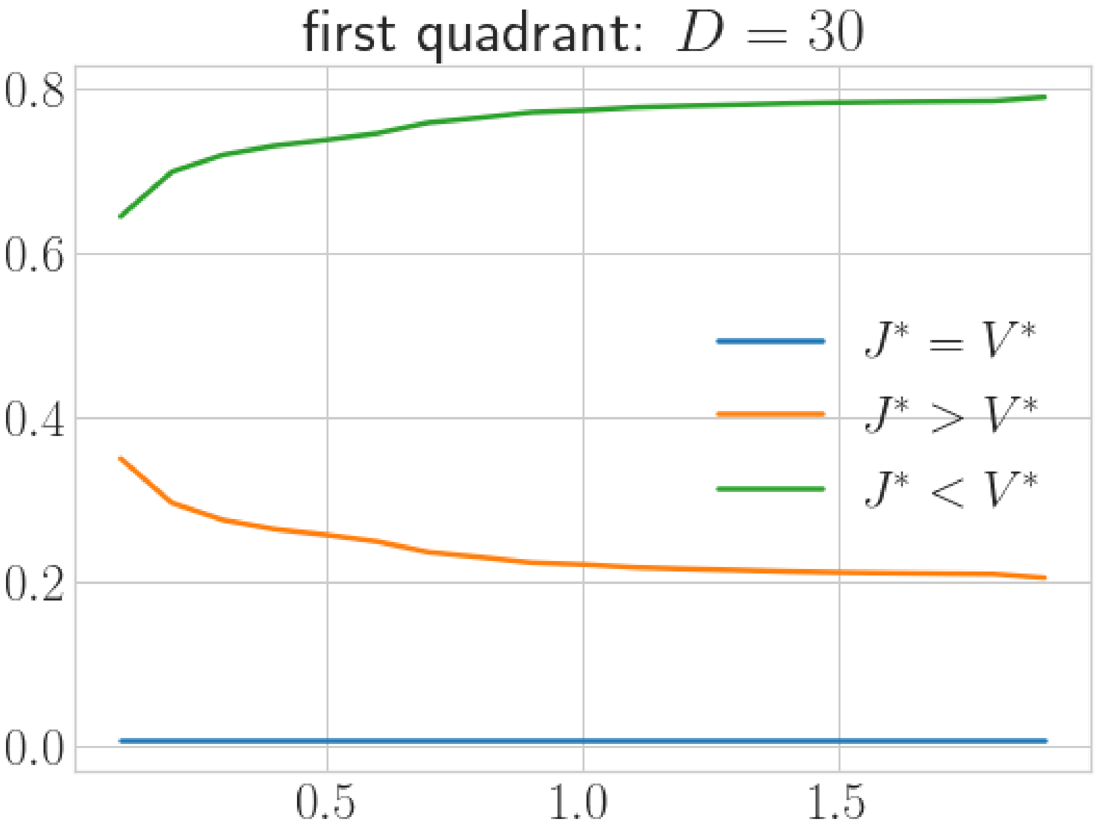
```
def plot_JvsV(D0_range, V0_range, J0_range, K0, sign, title):
    for D0 in D0_range:
        print (D0)
        for J0 in J0_range:
            C0, CJ, CV = [], [], []
            for V0 in V0_range:
                c0, cJ, cV = count_Vfp(D0, V0, J0, K0, sign)
                C0.append(c0/(c0+cJ+cV))
                CJ.append(cJ/(c0+cJ+cV))
                CV.append(cV/(c0+cJ+cV))
            plt.plot(V0_range, C0, label=r"$J^* = V^*$")
            plt.plot(V0_range, CJ, label=r"$J^* > V^*$")
            plt.plot(V0_range, CV, label=r"$J^* < V^*$")
            plt.legend()
            plt.title(title+r"$D={}, J={}$".format(D0, J0))
            plt.show()
```

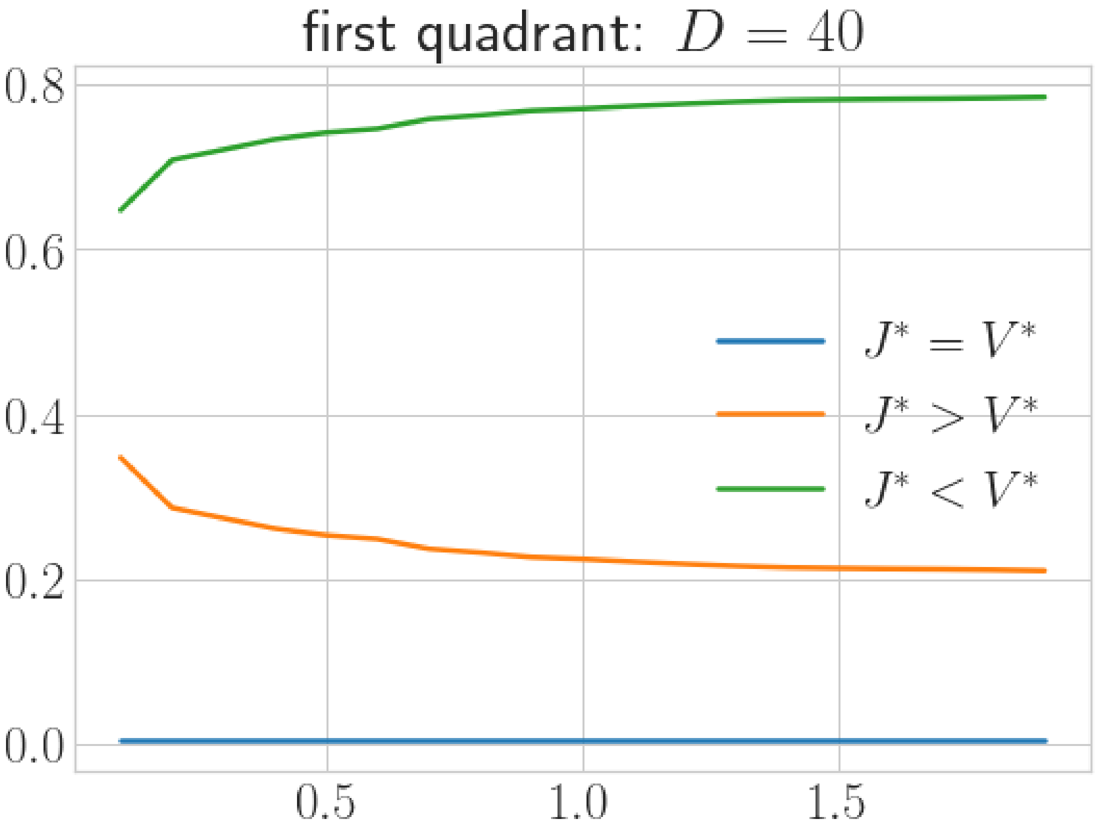
[18]:

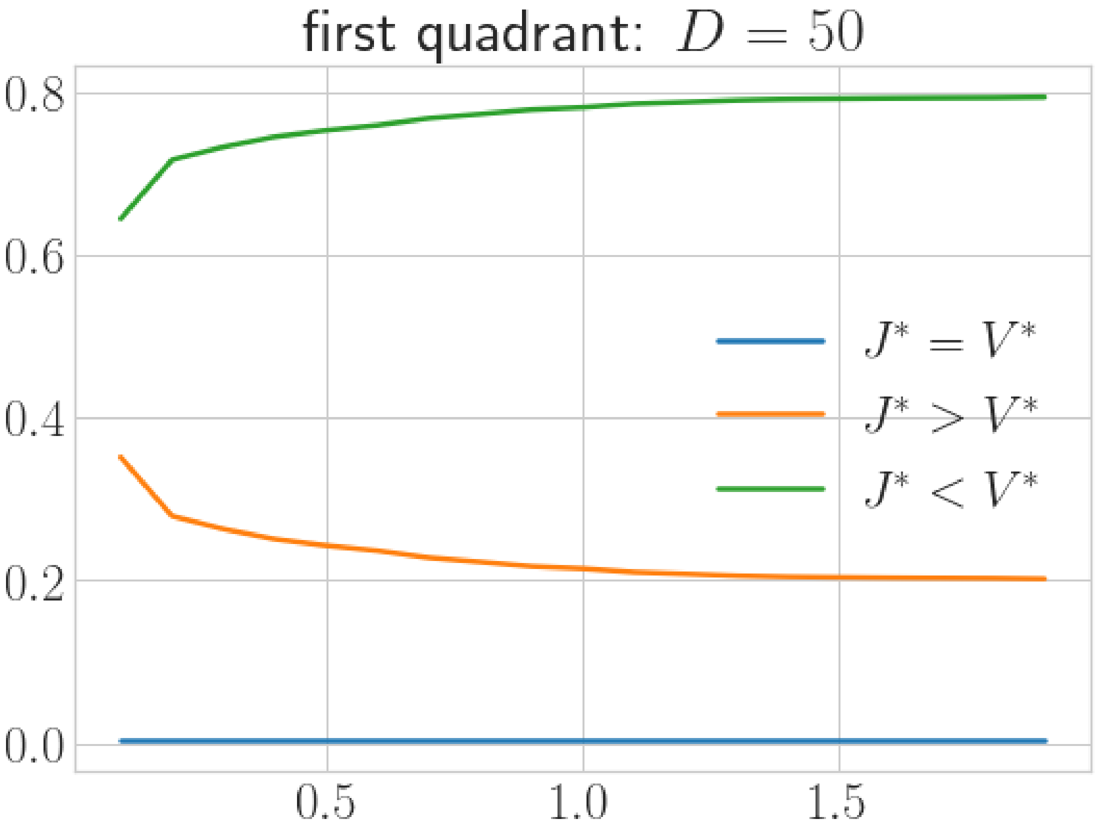
```
plot_JvsV(range(10,60,30), np.arange(0.1,2,0.1), 0.5, 0.4, 1, r'first quadrant:␣
 ↪')
```

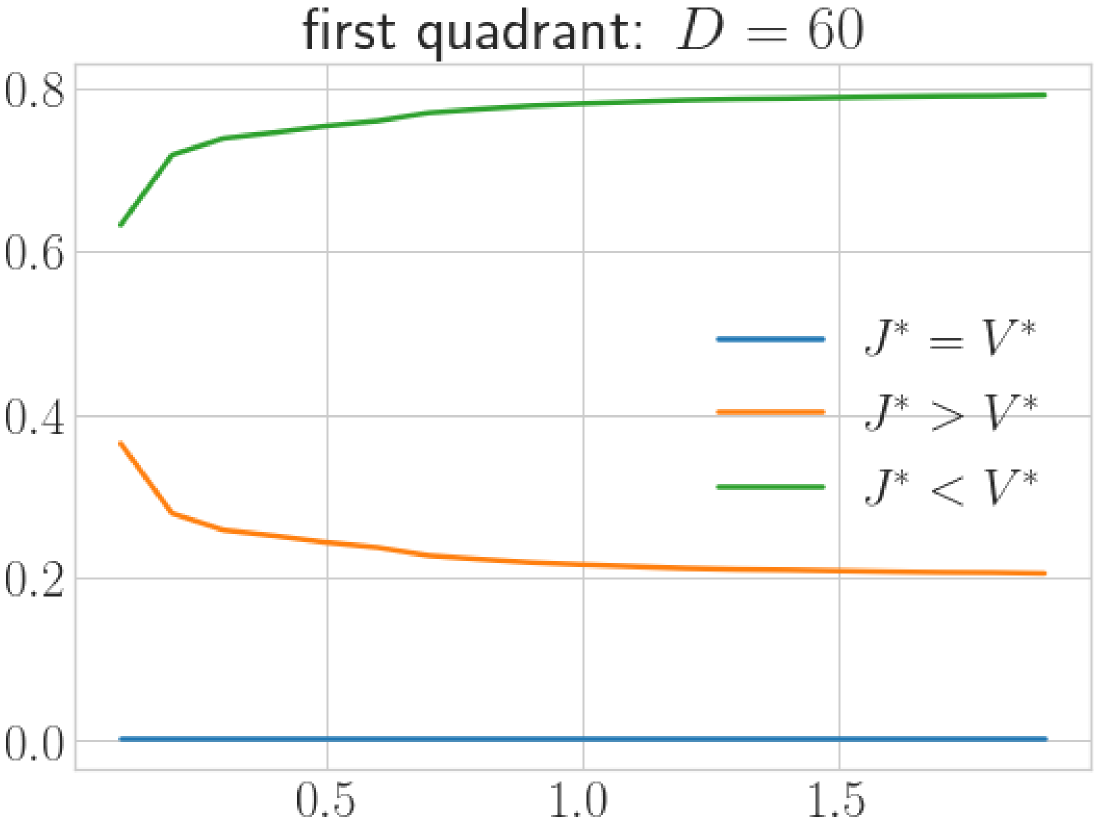


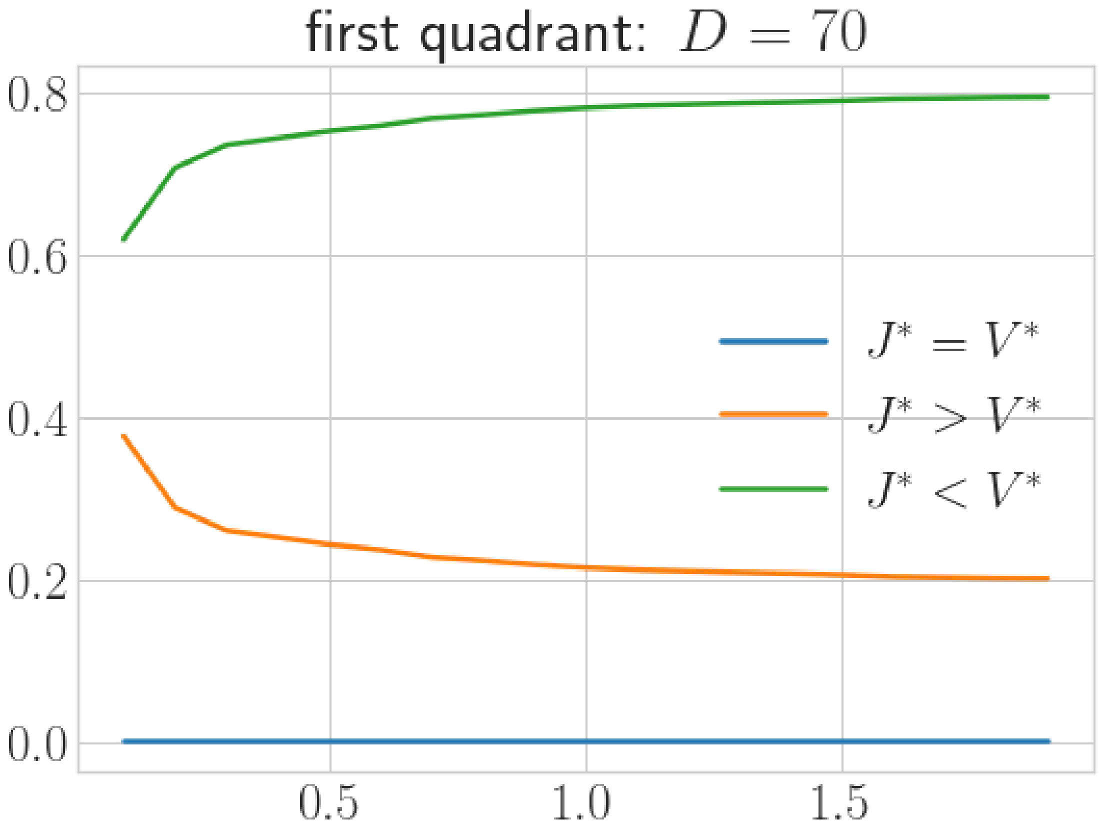

first quadrant: $D = 20$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

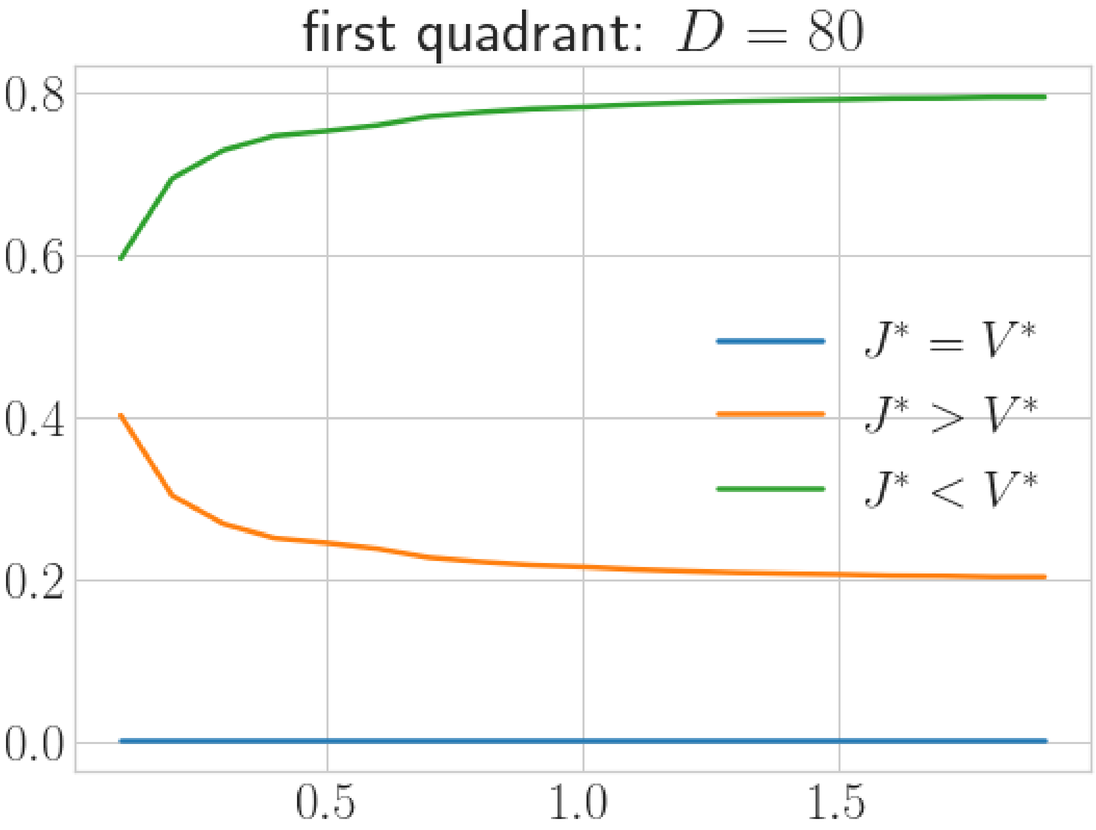

first quadrant: $D = 30$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

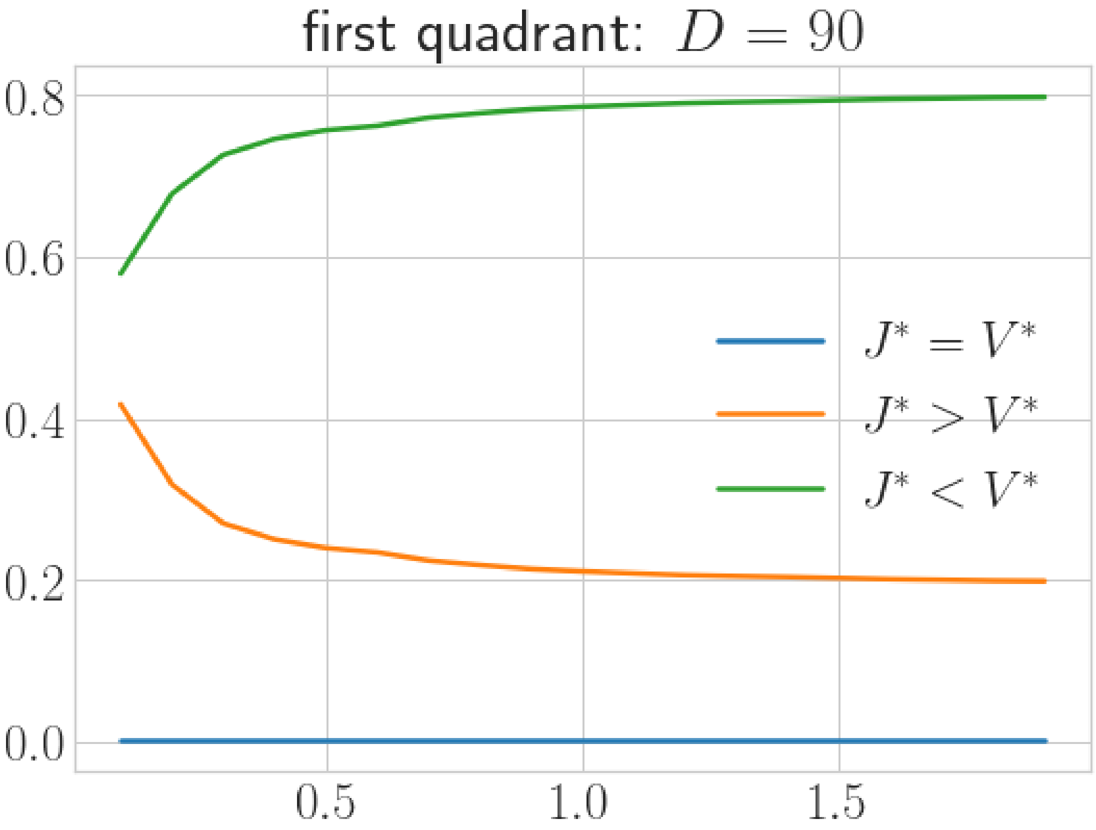

first quadrant: $D = 40$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

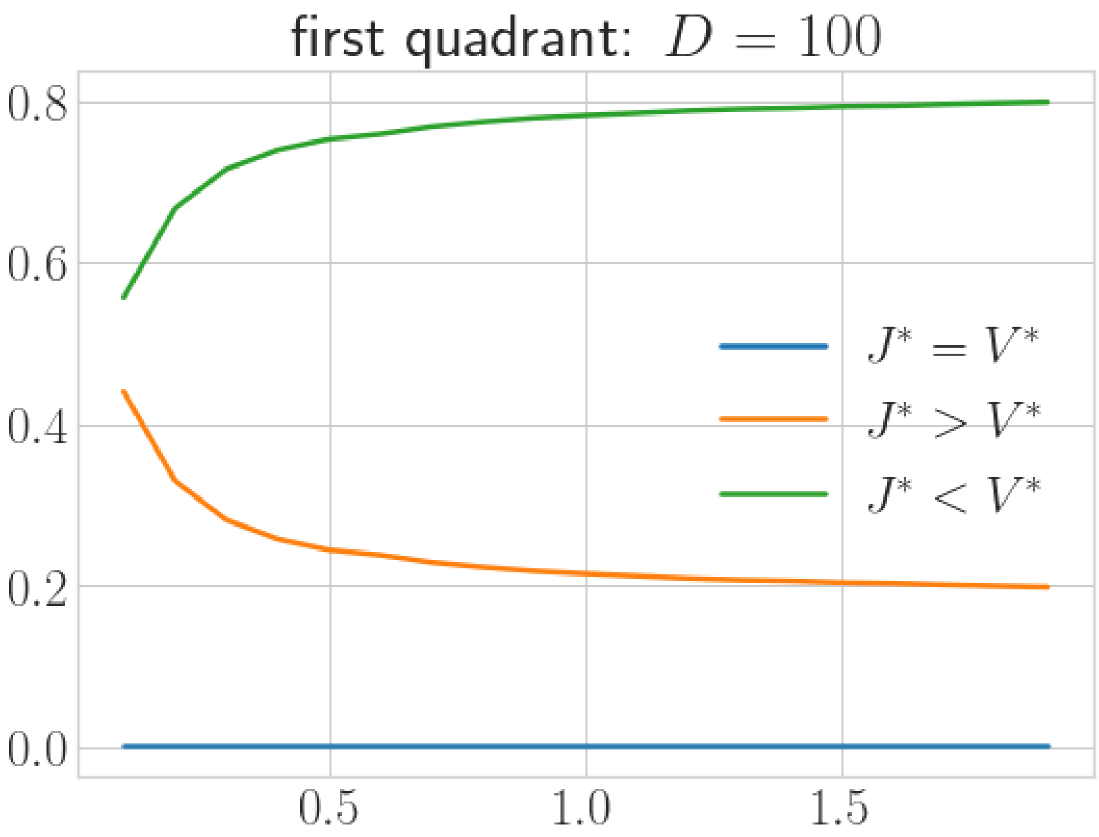

first quadrant: $D = 50$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$


first quadrant: $D = 60$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$


first quadrant: $D = 70$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

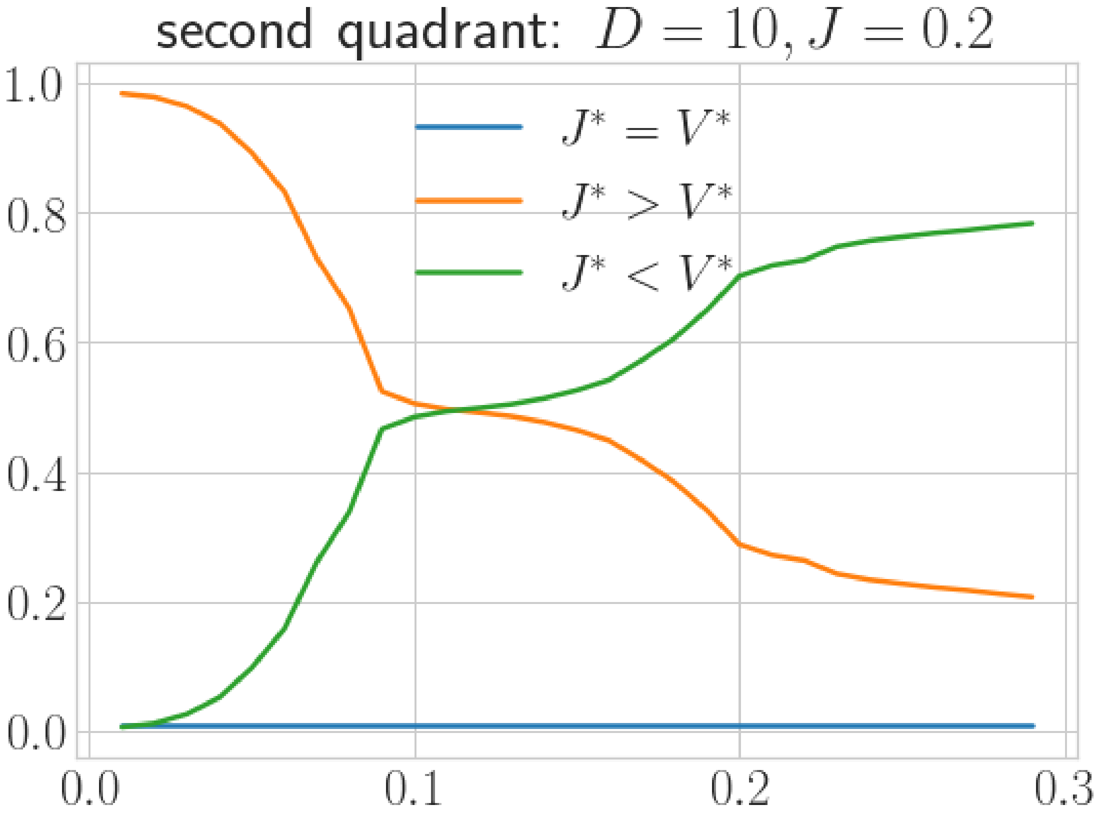

first quadrant: $D = 80$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

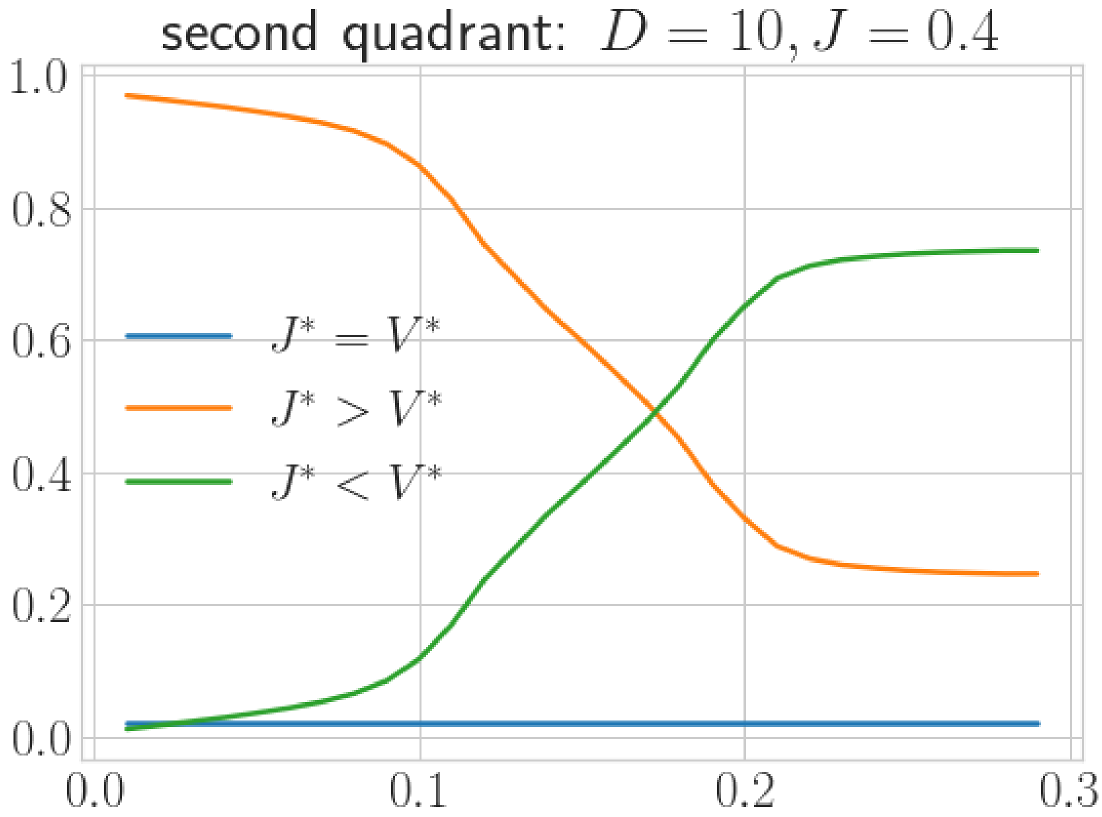

first quadrant: $D = 90$
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$



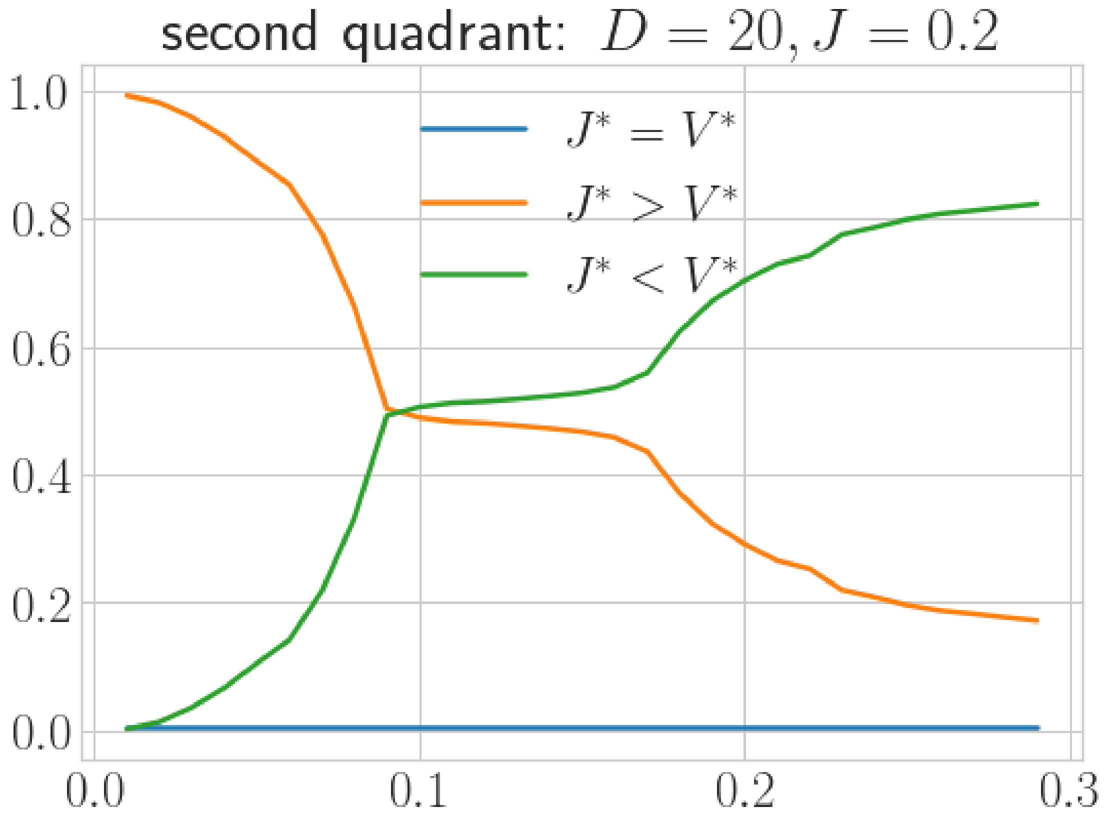
```
[21]: plot_JvsV(range(10,31,10), np.arange(0.01,0.3,0.01), [0.2, 0.4], 0.5, 1,␣
       ↪r'second quadrant: ')
```

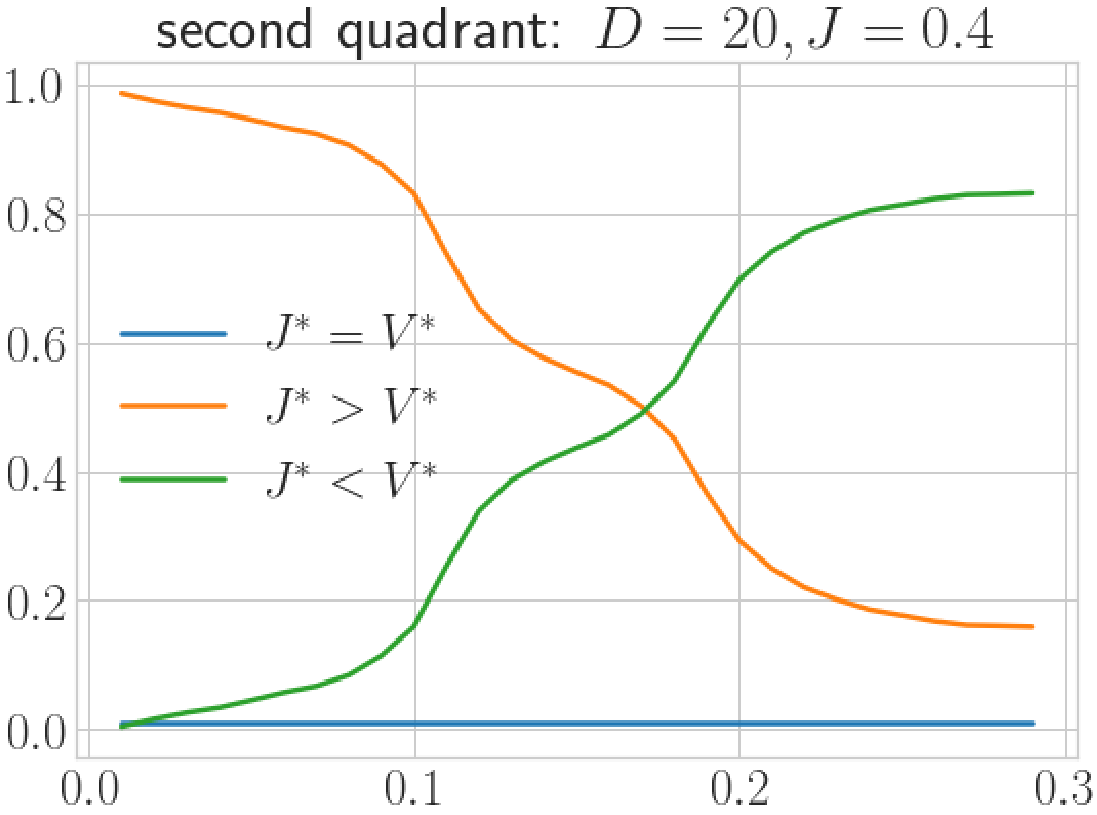
```
10
```


second quadrant: $D = 10, J = 0.2$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

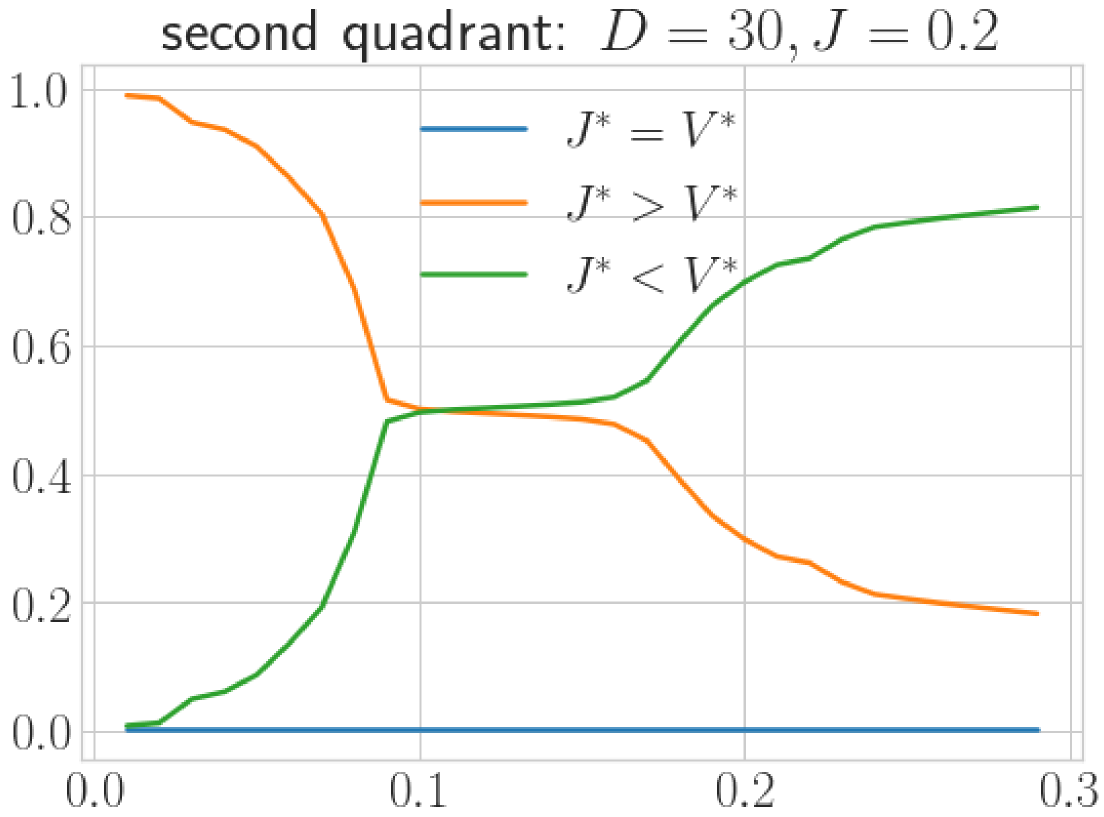


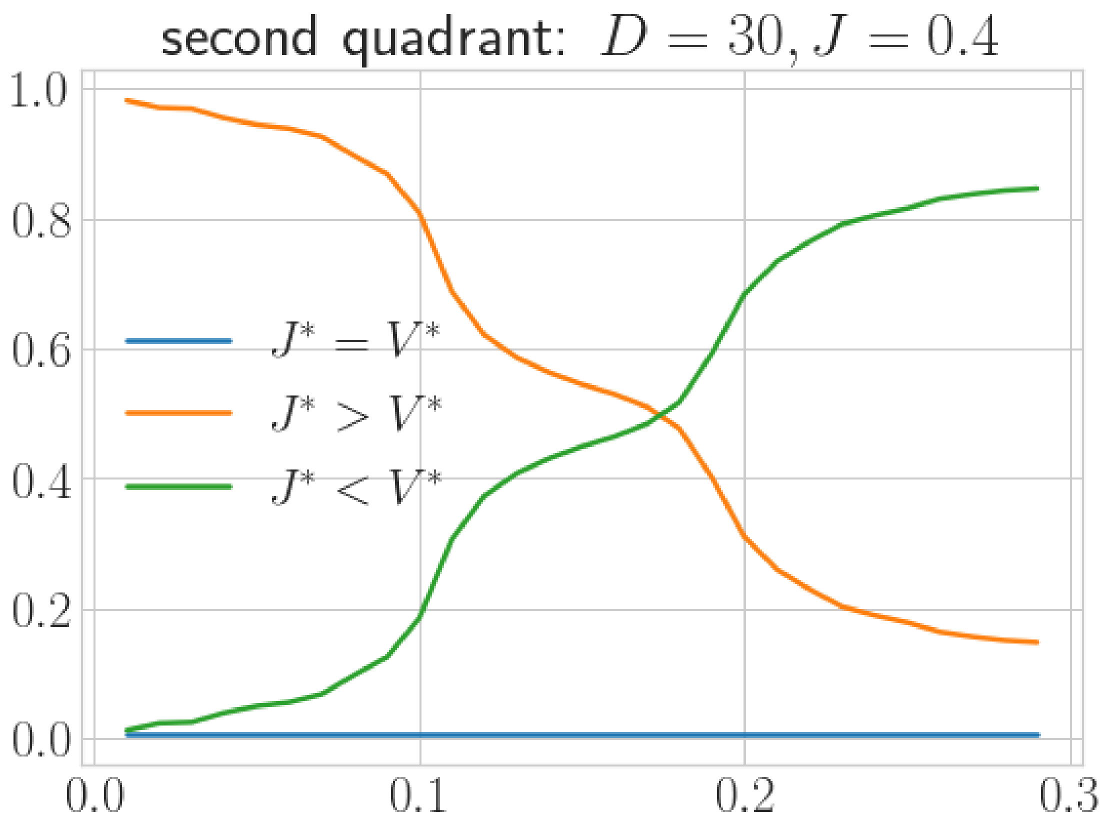
20


second quadrant: $D = 20, J = 0.2$
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3



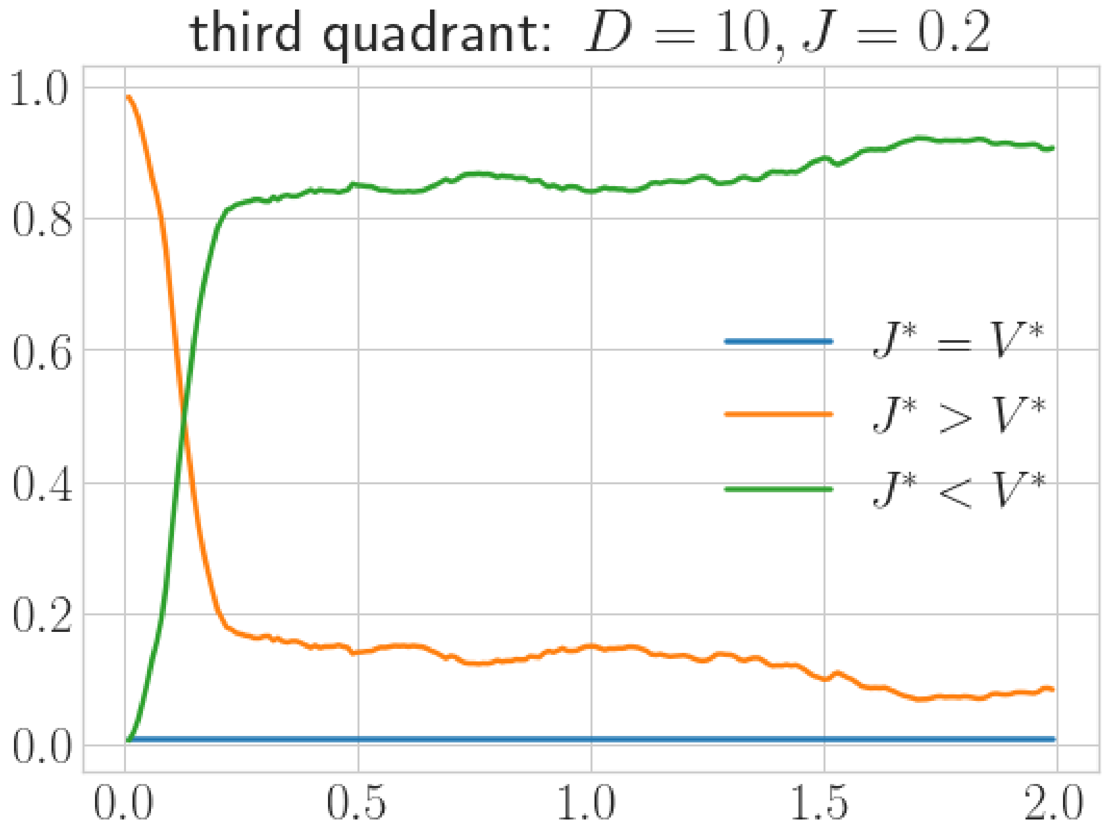
30

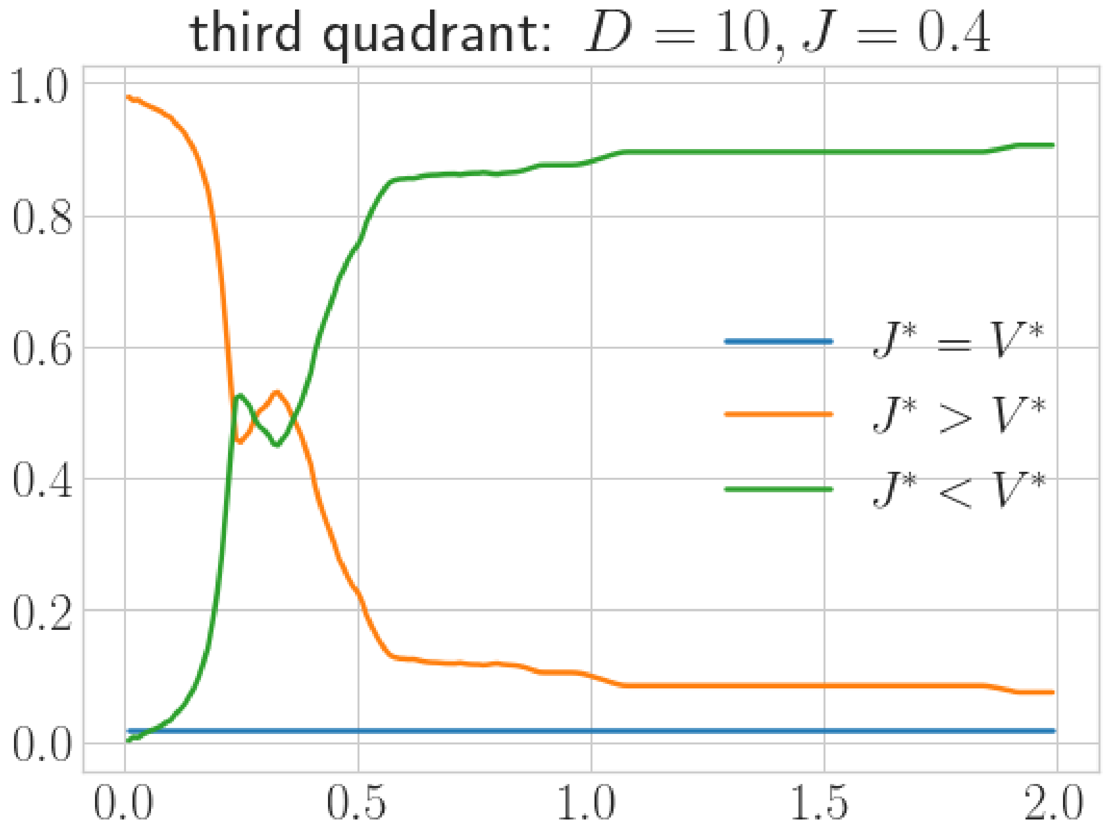

second quadrant: $D = 30, J = 0.2$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$



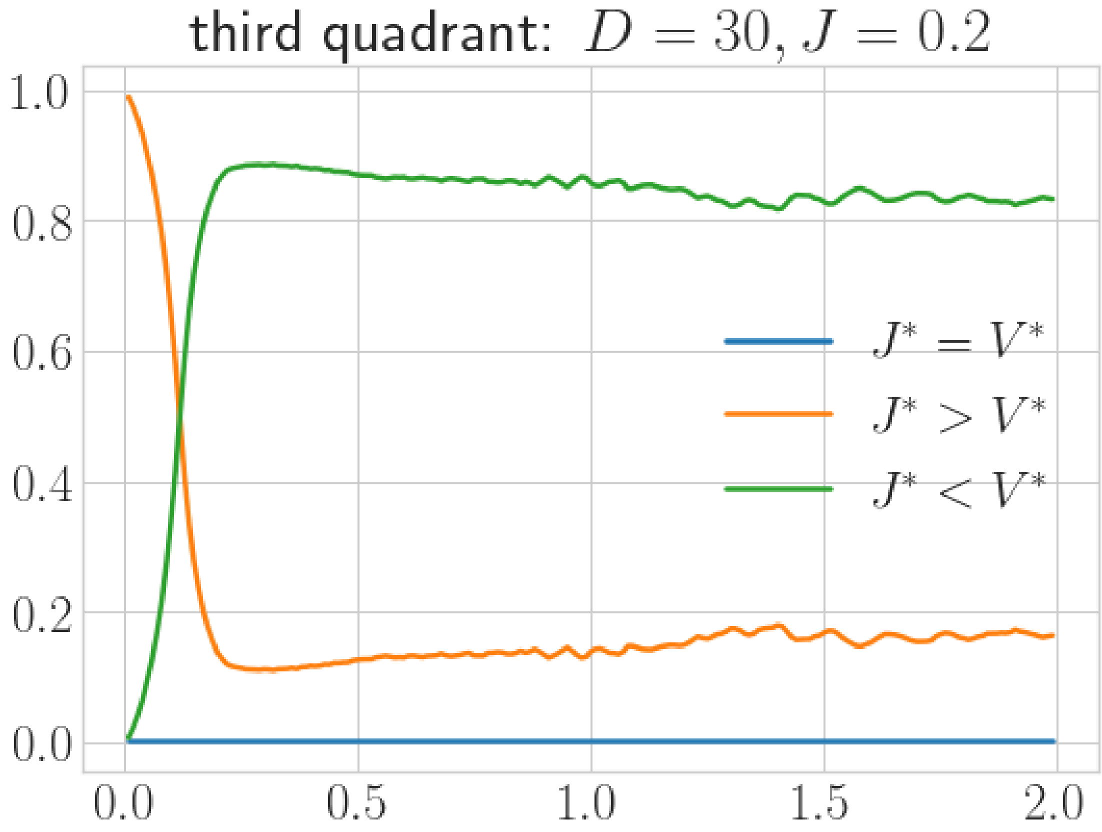
```
[22]: plot_JvsV([10, 30, 60], np.arange(0.01,2,0.01), [0.2, 0.4], 0.5, -1, r'third␣
       ↪quadrant: ')
```

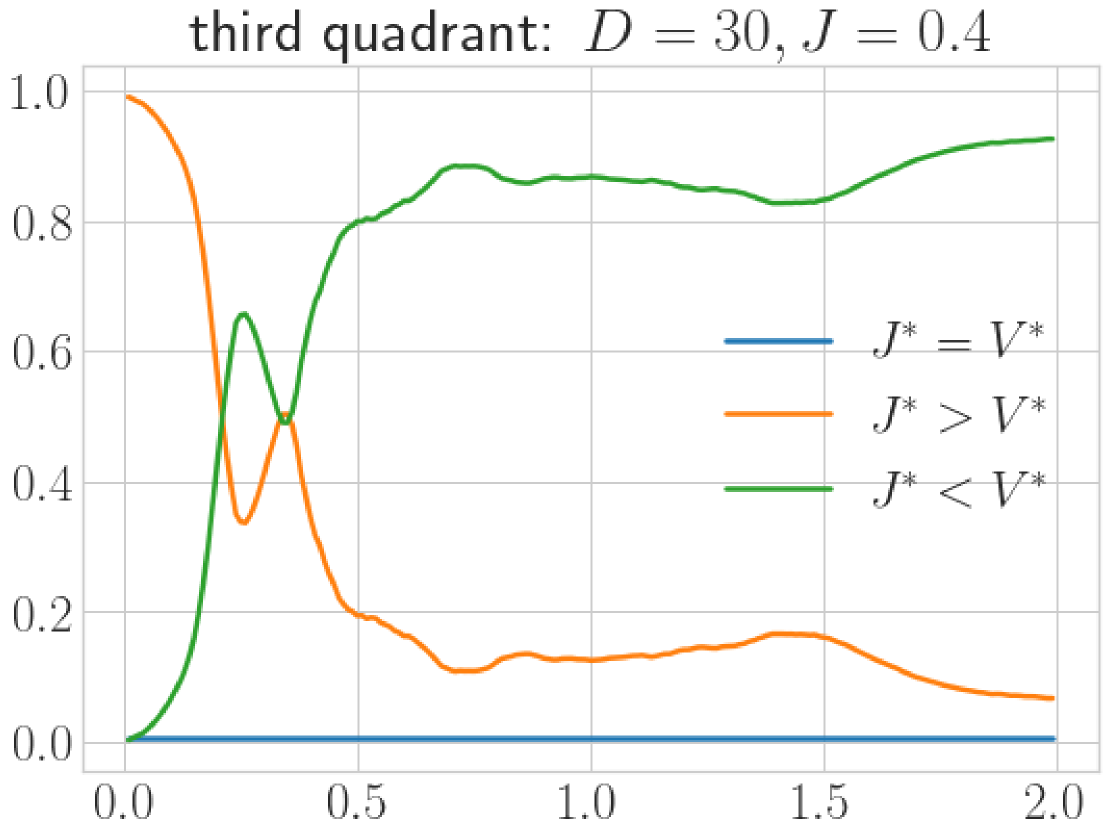
```
10
```


third quadrant: $D = 10, J = 0.2$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

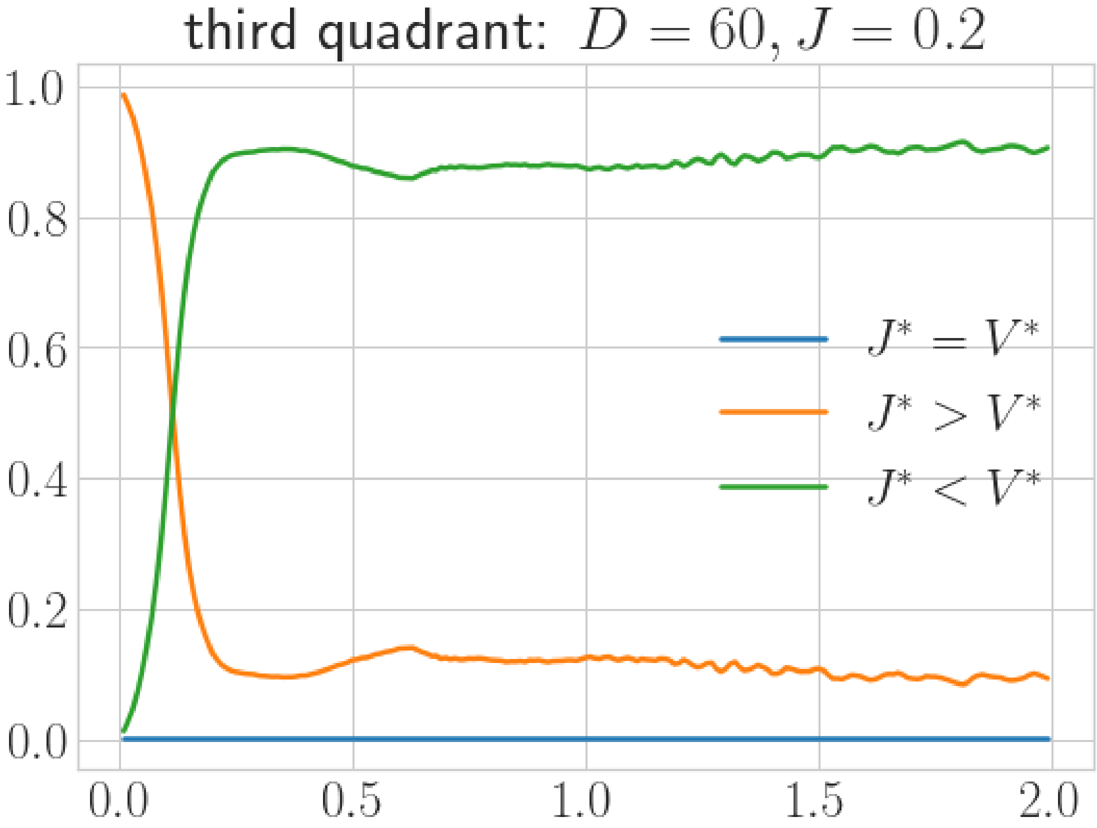


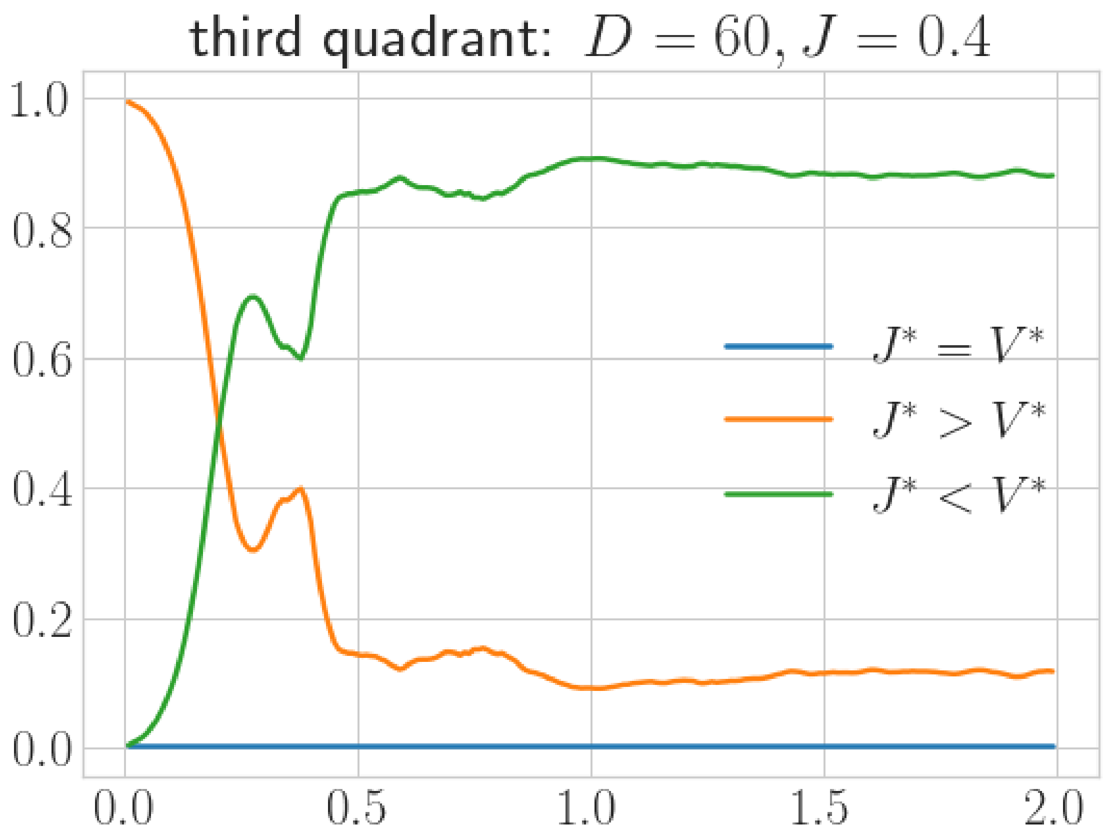
30


third quadrant: $D = 30, J = 0.2$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$



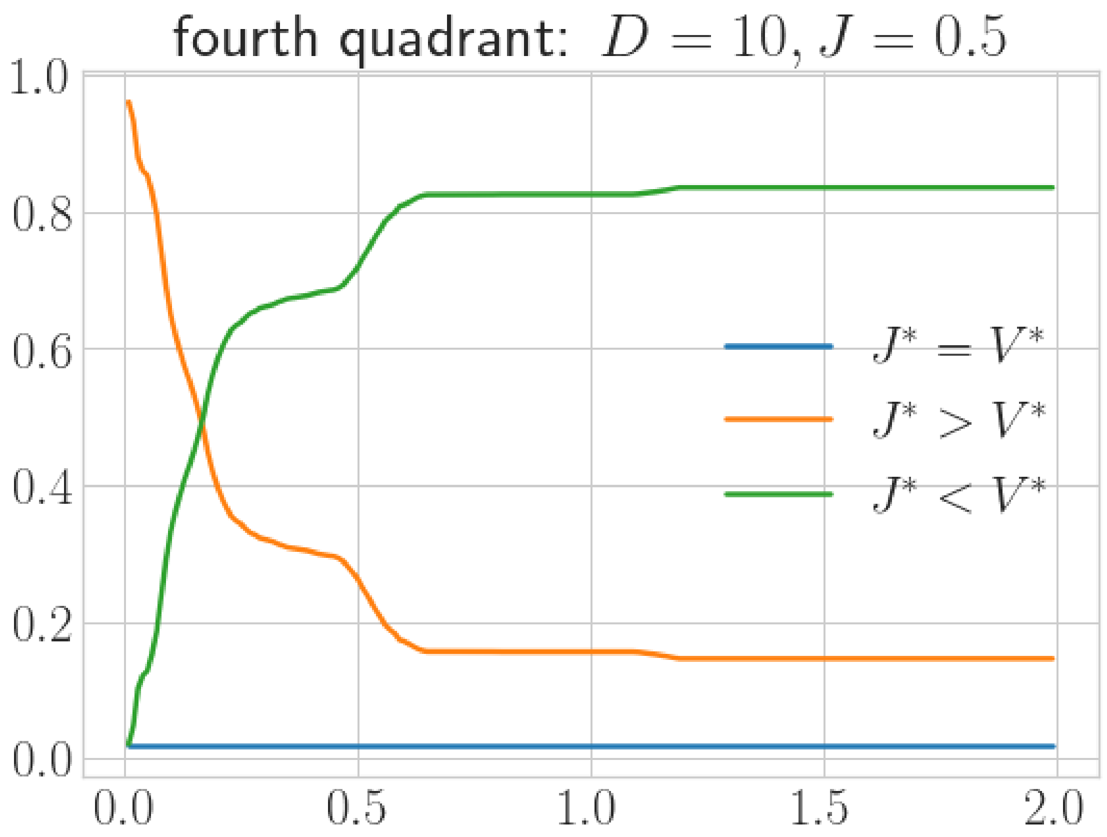
60

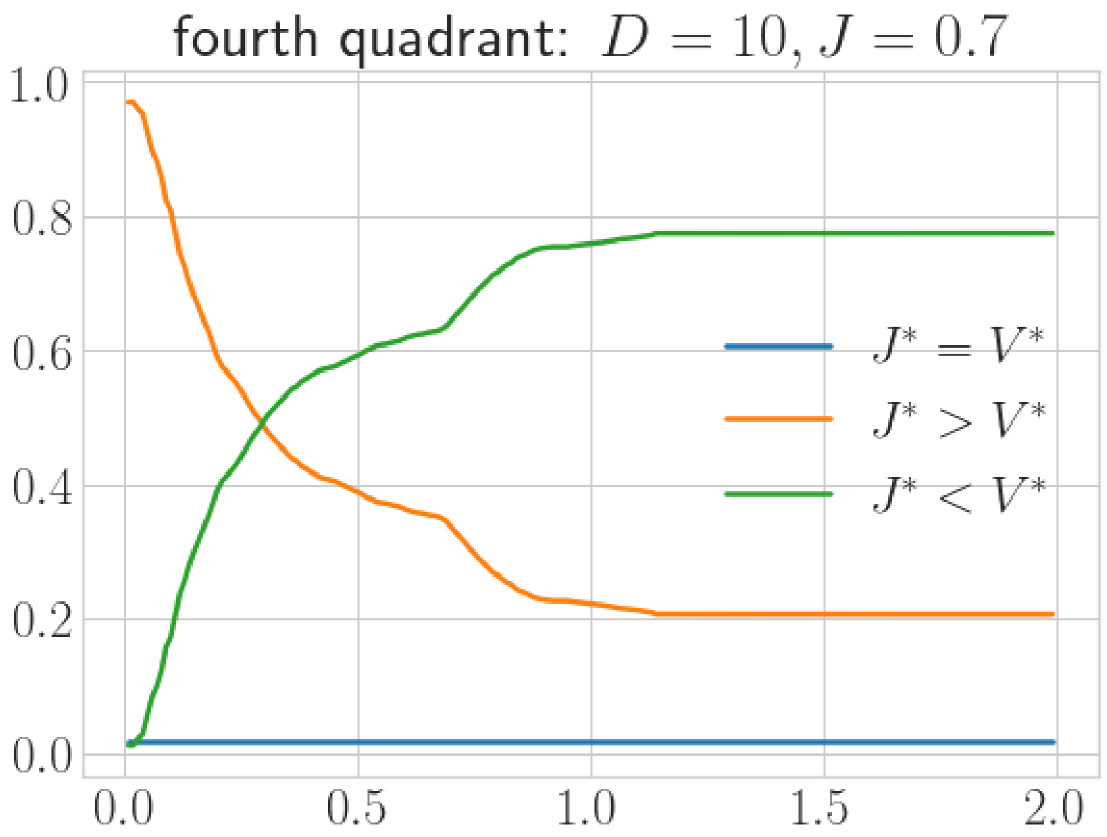

third quadrant: $D = 60, J = 0.2$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$



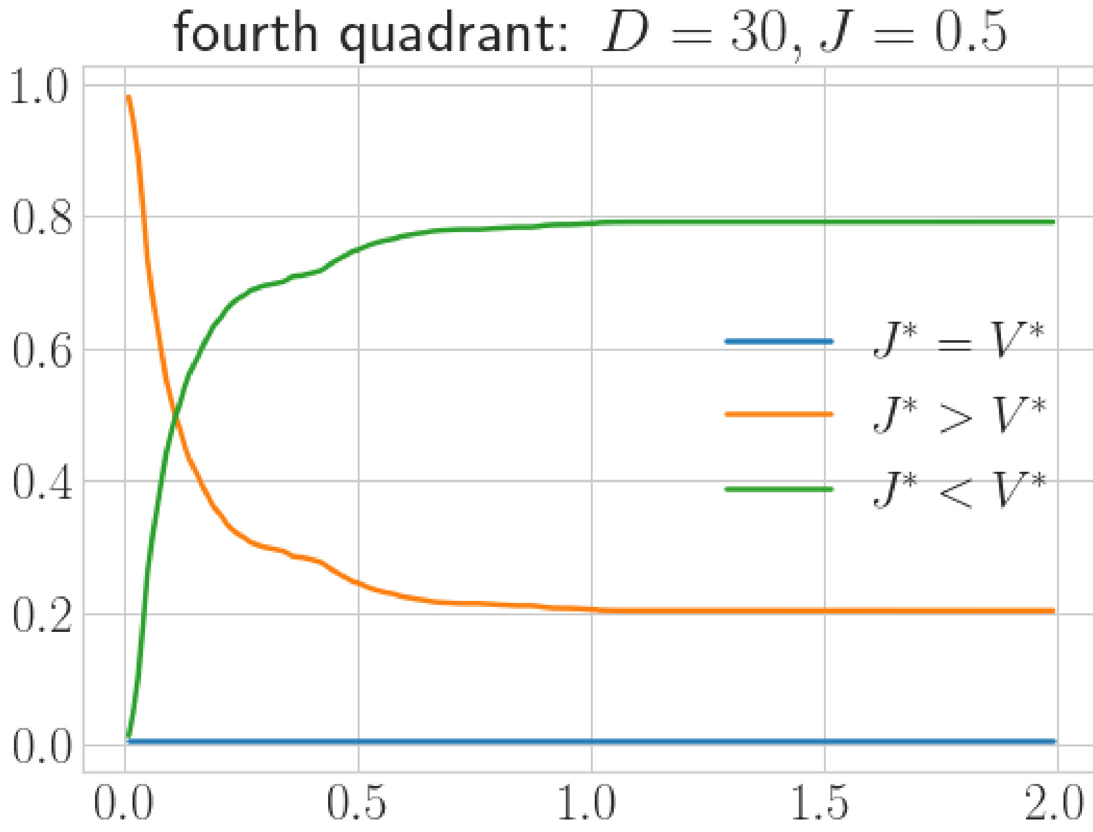
```
[23]: plot_JvsV([10, 30, 60], np.arange(0.01,2,0.01), [0.5, 0.7], 0.4, -1, r'fourth␣
       ↪quadrant: ')
```

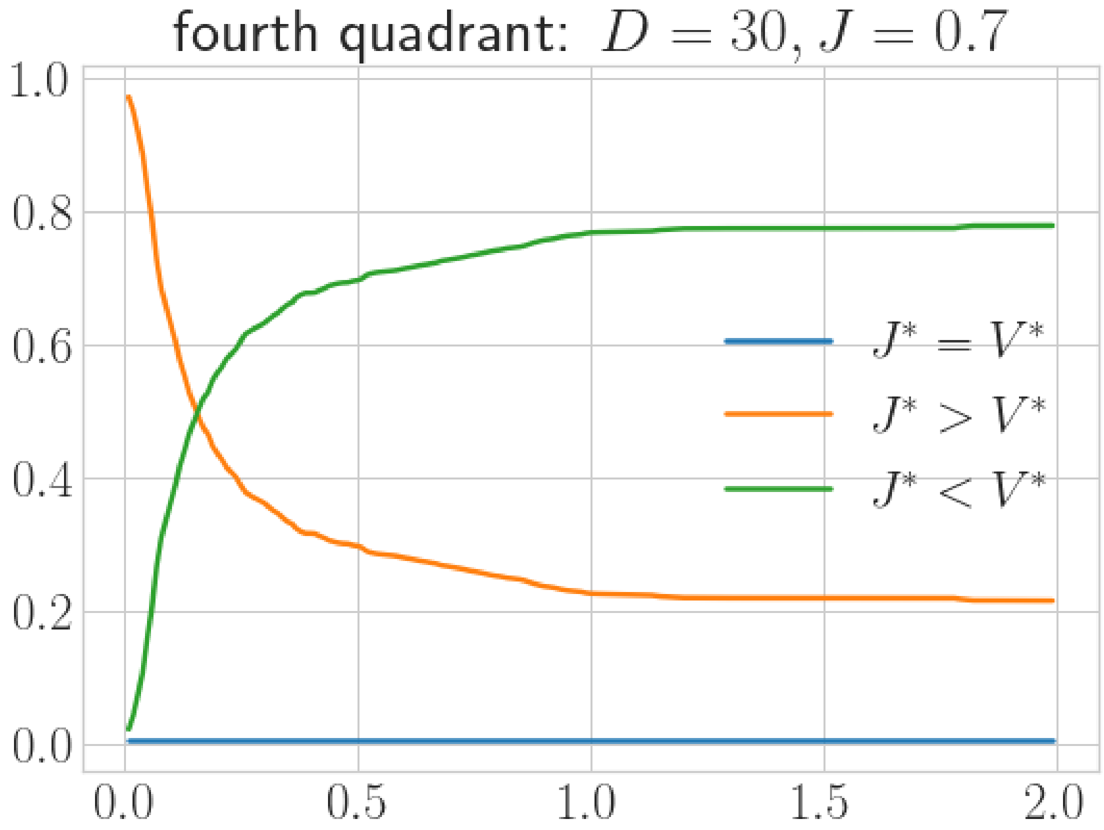
```
10
```


fourth quadrant: $D = 10, J = 0.5$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

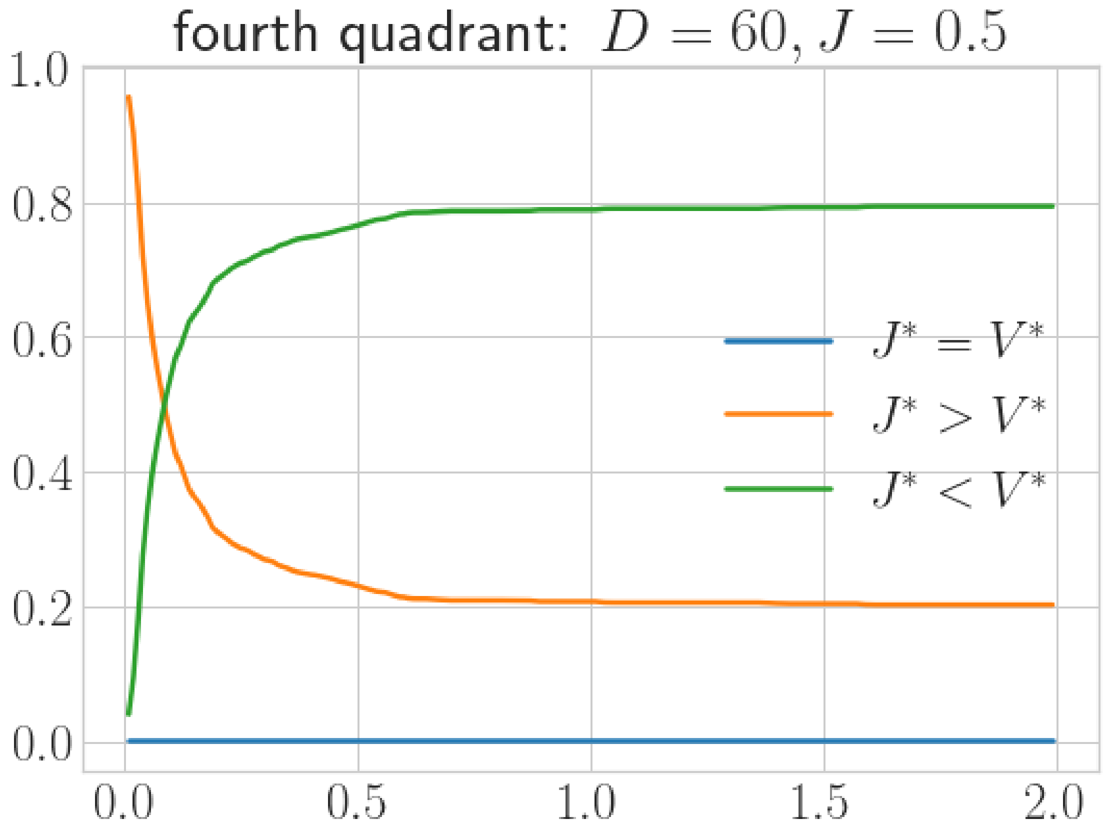


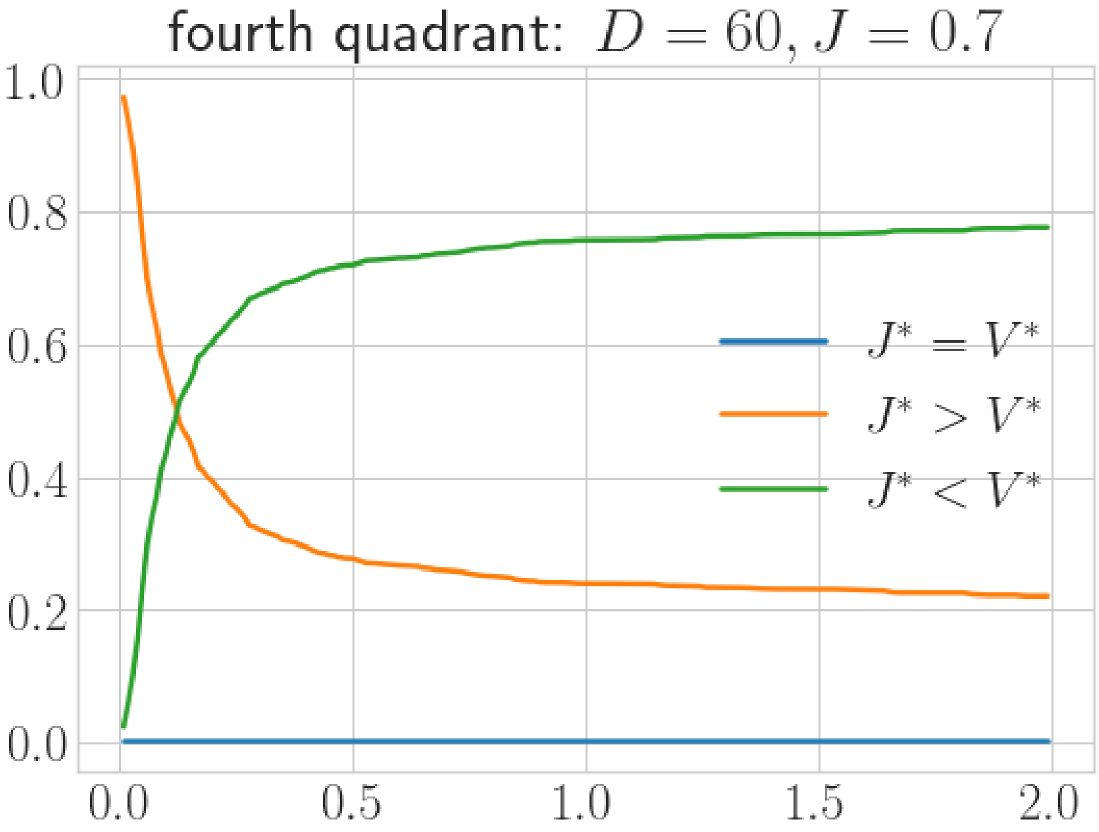
30


fourth quadrant: $D = 30, J = 0.5$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

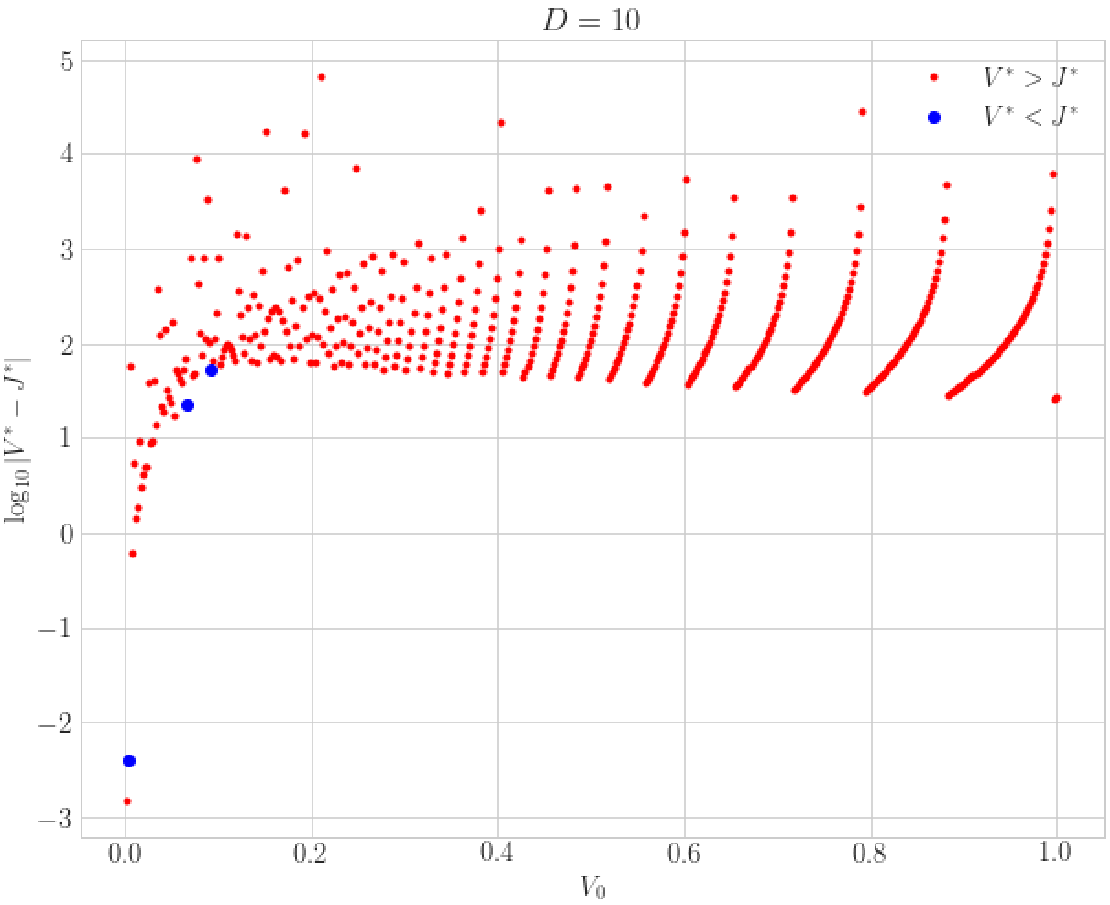


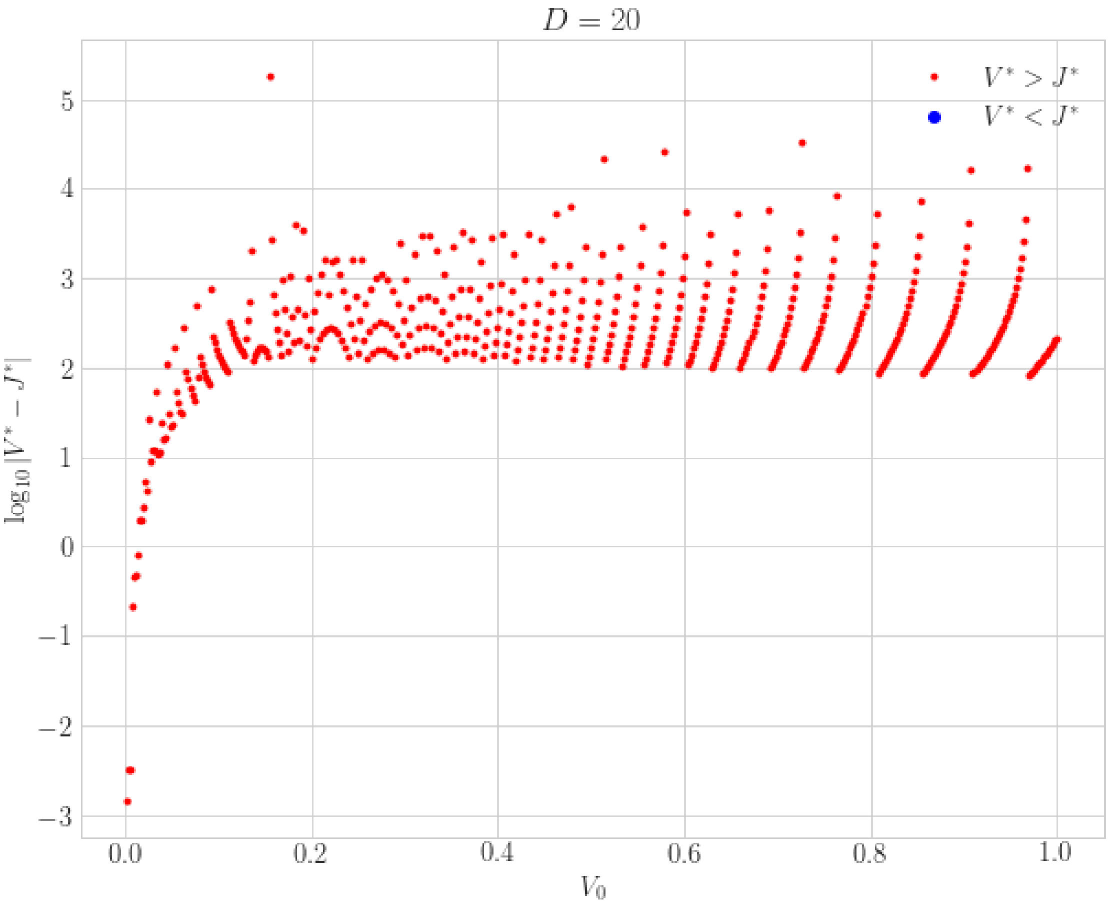
60


fourth quadrant: $D = 60$, $J = 0.5$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
$J^* = V^*$
$J^* > V^*$
$J^* < V^*$

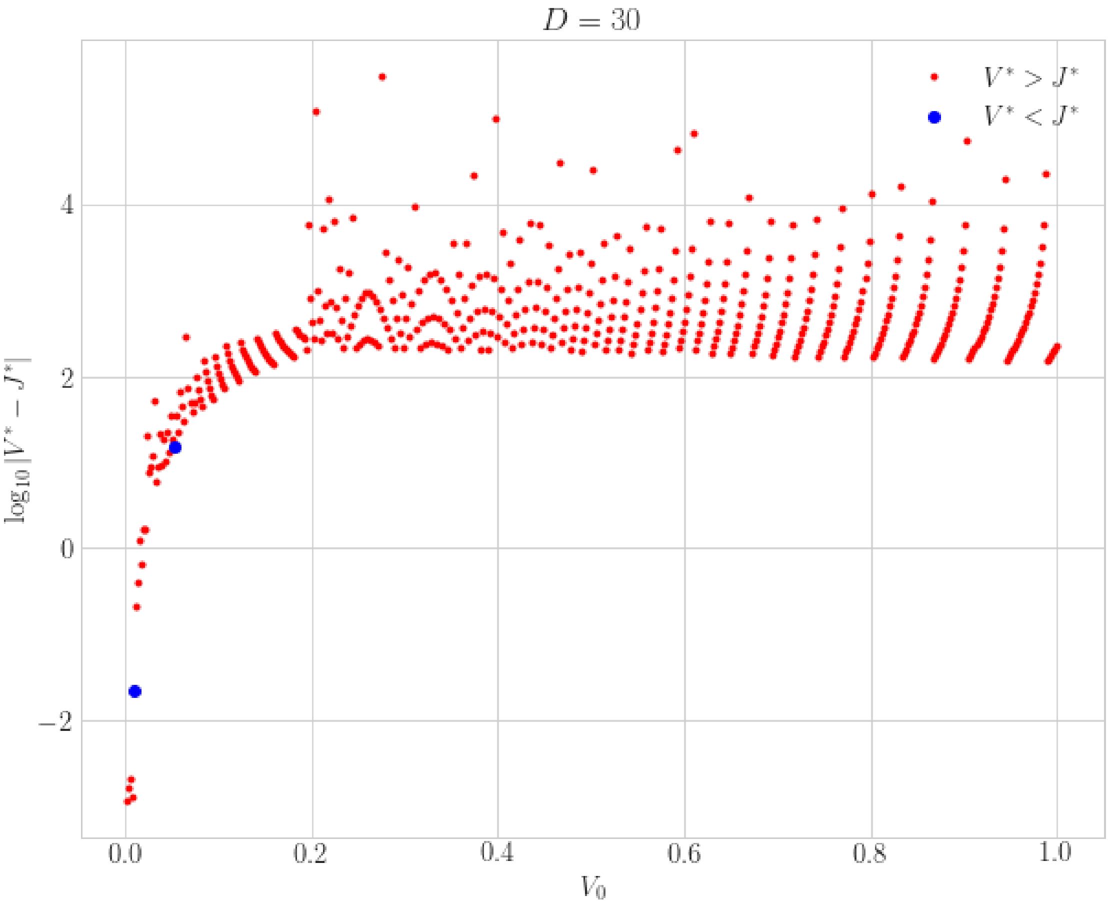


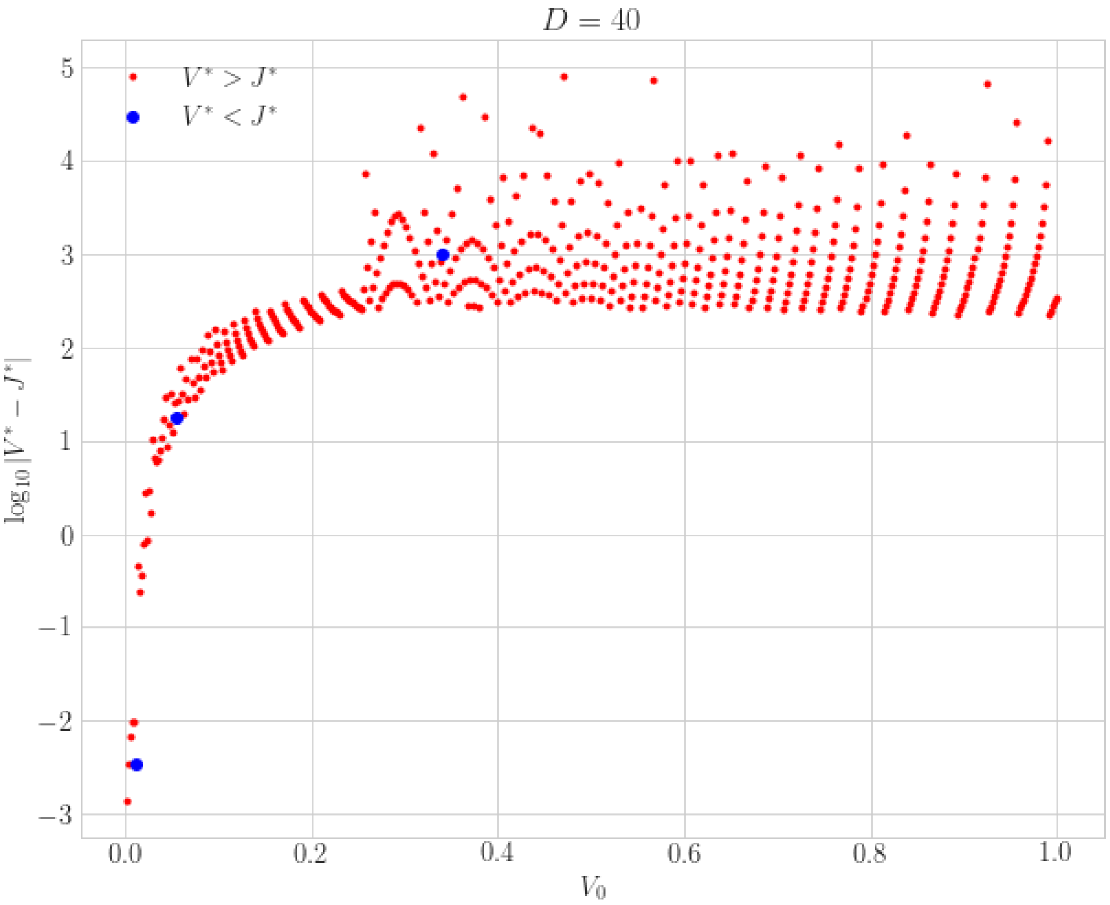
[22]:
```
plot_JvsV(-1, 1, r'second quadrant')
```

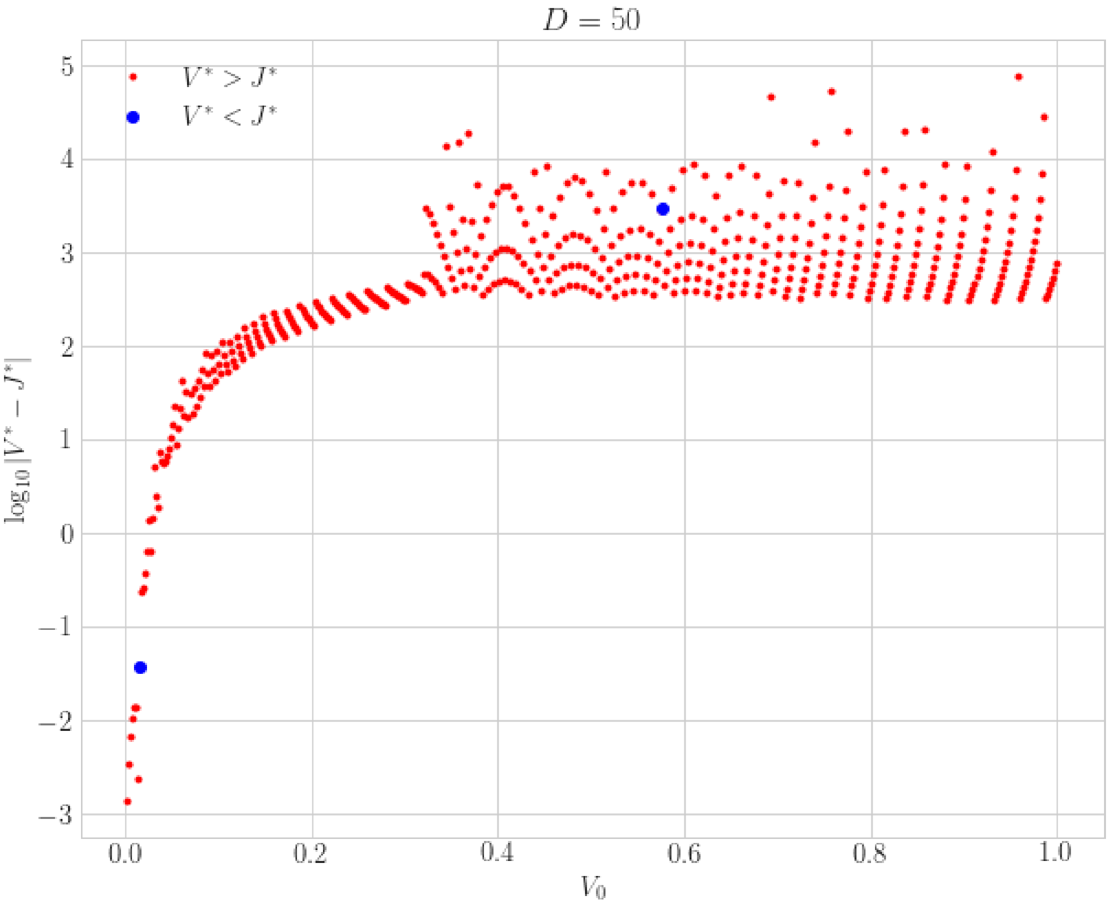

$D = 10$
$V^* > J^*$
$V^* < J^*$
$\log_{10}|V^* - J^*|$
5
4
3
2
1
0
-1
-2
-3
0.0
0.2
0.4
0.6
0.8
1.0
V_0



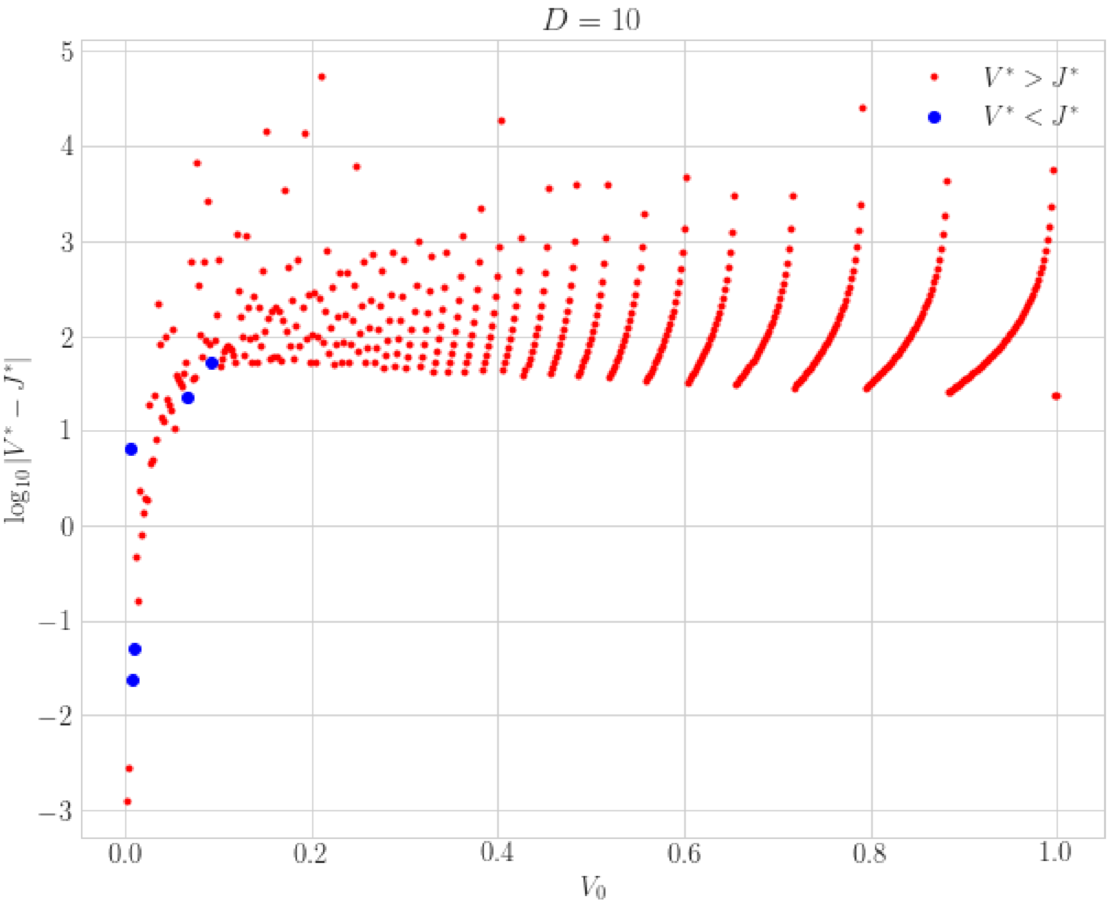
$D = 20$
$V^* > J^*$
$V^* < J^*$
$\log_{10}|V^* - J^*|$
5
4
3
2
1
0
-1
-2
-3
0.0
0.2
0.4
0.6
0.8
1.0
V_0

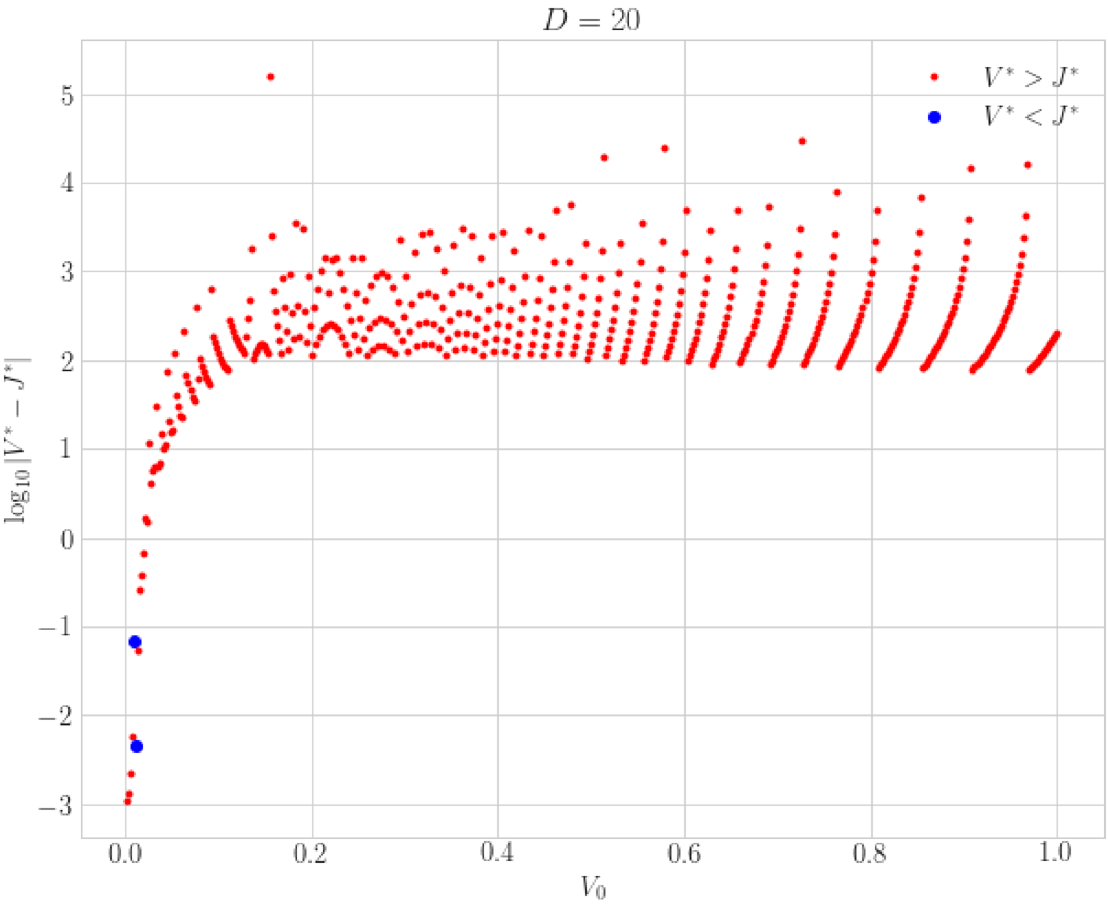

$D = 30$
$V^* > J^*$
$V^* < J^*$
$\log_{10}|V^* - J^*|$
V_0
4
2
0
−2
0.0
0.2
0.4
0.6
0.8
1.0


$D = 40$
$V^* > J^*$
$V^* < J^*$
$\log_{10}|V^* - J^*|$
5
4
3
2
1
0
-1
-2
-3
0.0
0.2
0.4
0.6
0.8
1.0
V_0



```
[23]: plot_JvsV(-1, -1, r'third quadrant')
```


$D = 10$
$\log_{10}|V^* - J^*|$
5
4
3
2
1
0
-1
-2
-3
0.0
0.2
0.4
0.6
0.8
1.0
V_0
● $V^* > J^*$
● $V^* < J^*$



$D = 20$

$\log_{10}|V^* - J^*|$

V_0

● $V^* > J^*$

● $V^* < J^*$

5

4

3

2

1

0

−1

−2

−3

0.0

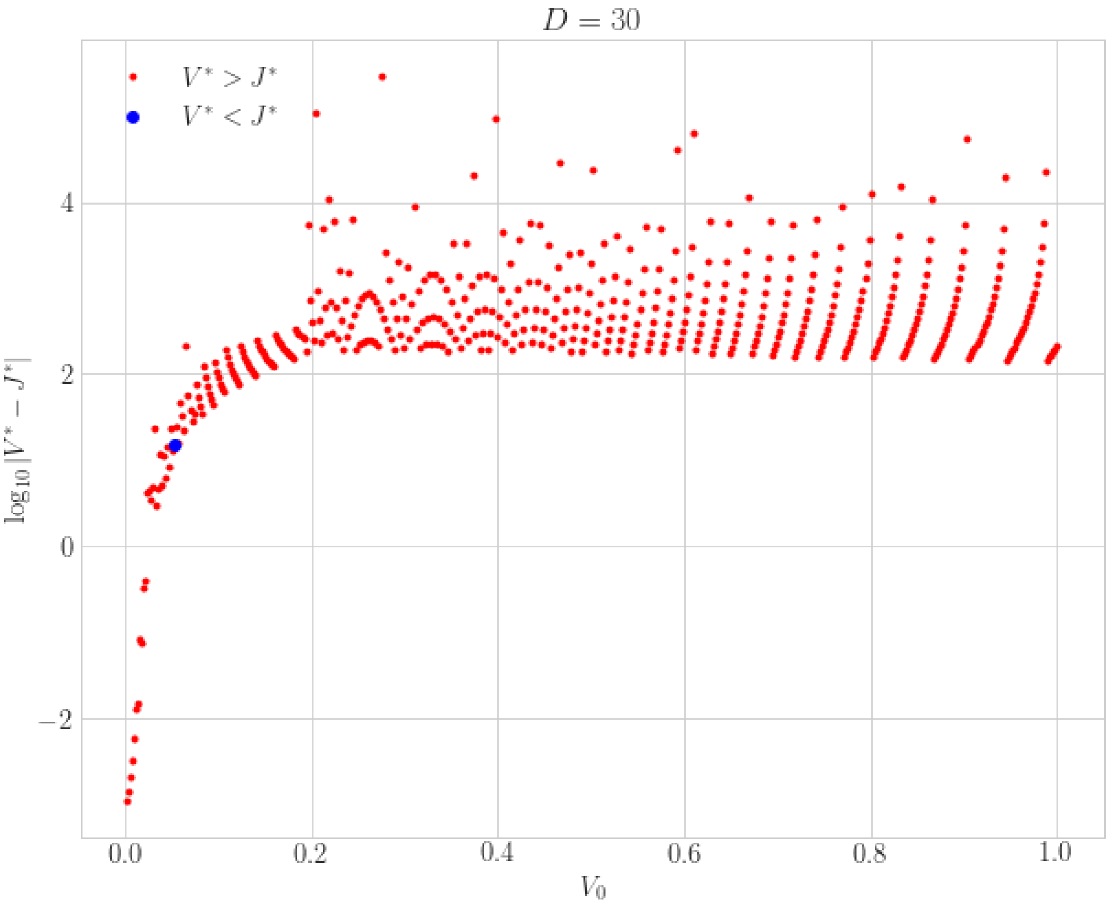
0.2

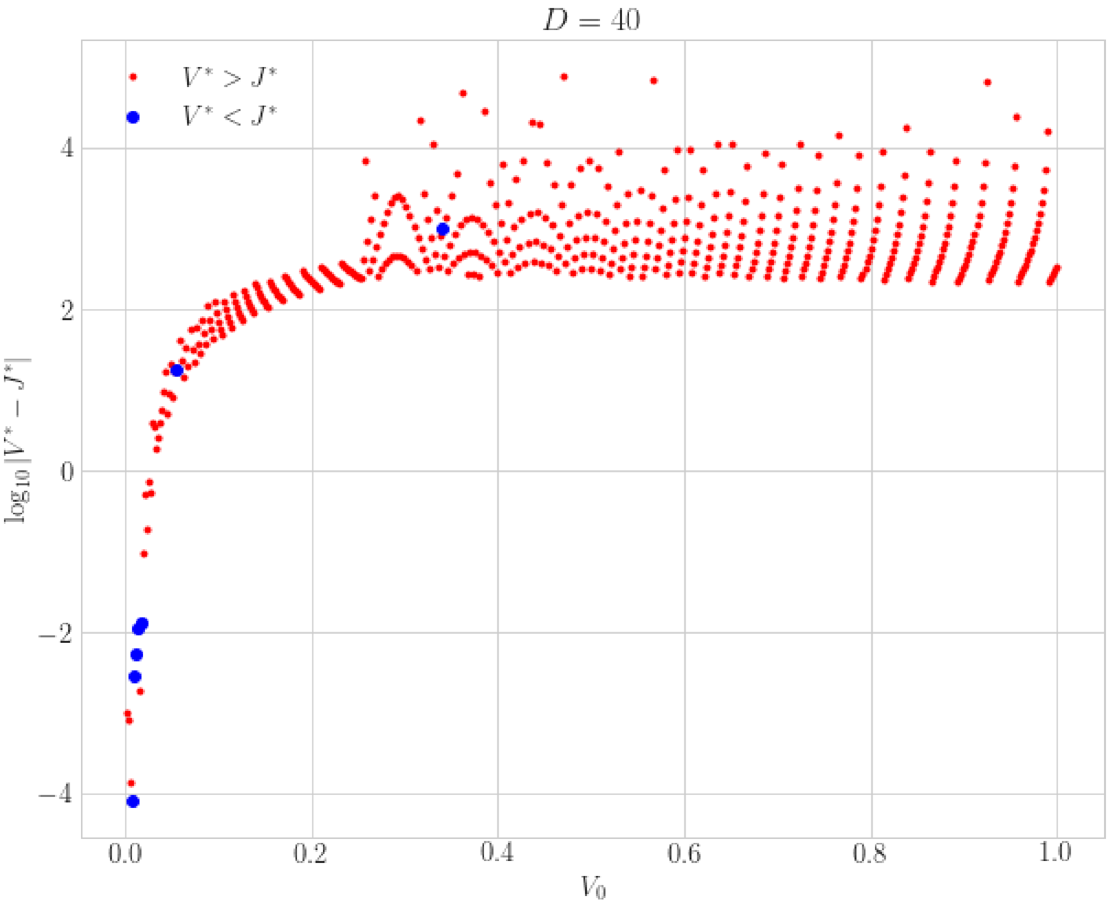
0.4

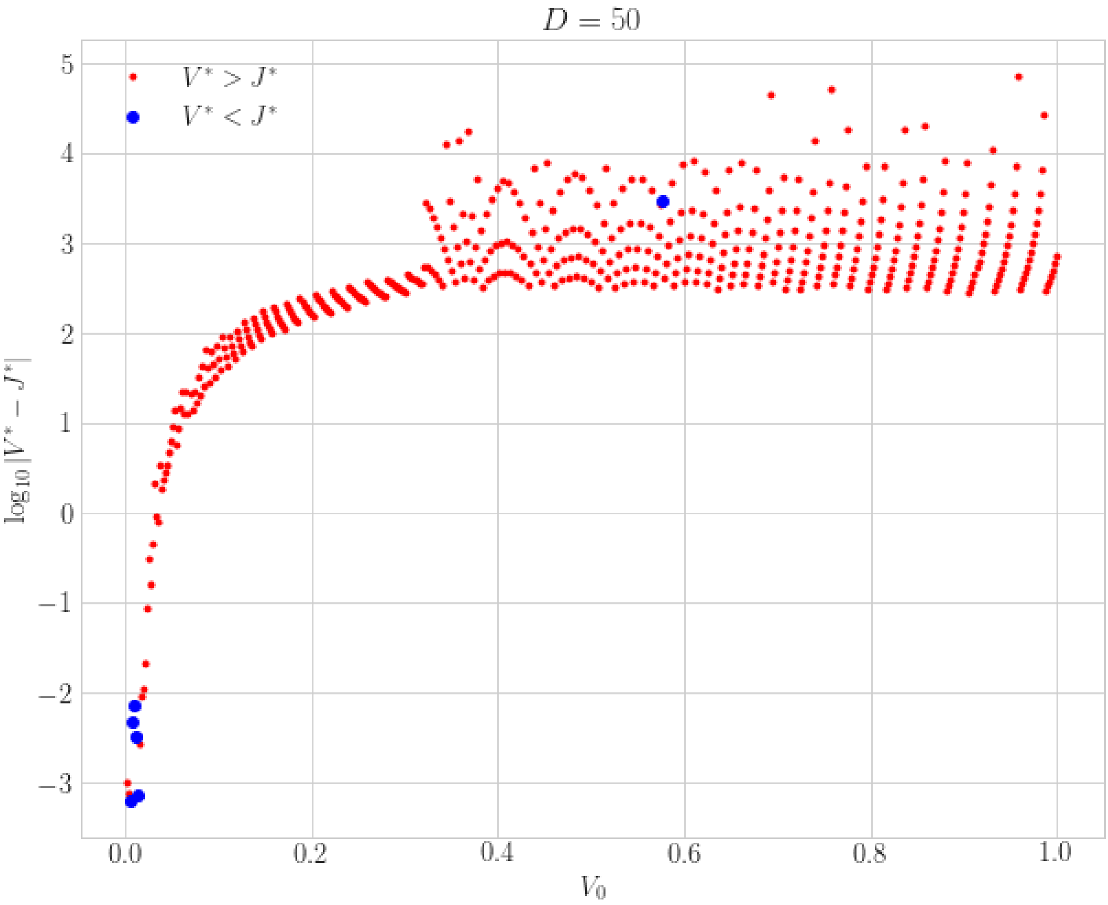
0.6

0.8

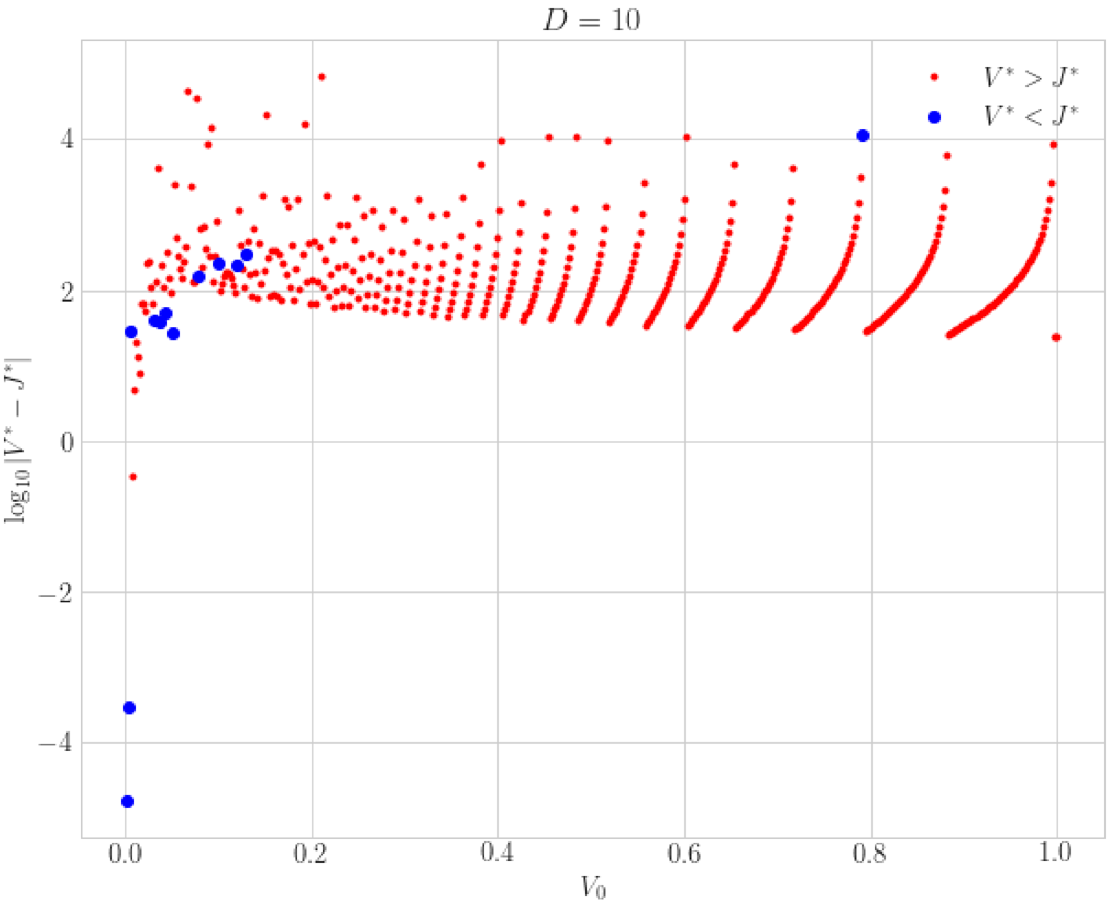
1.0

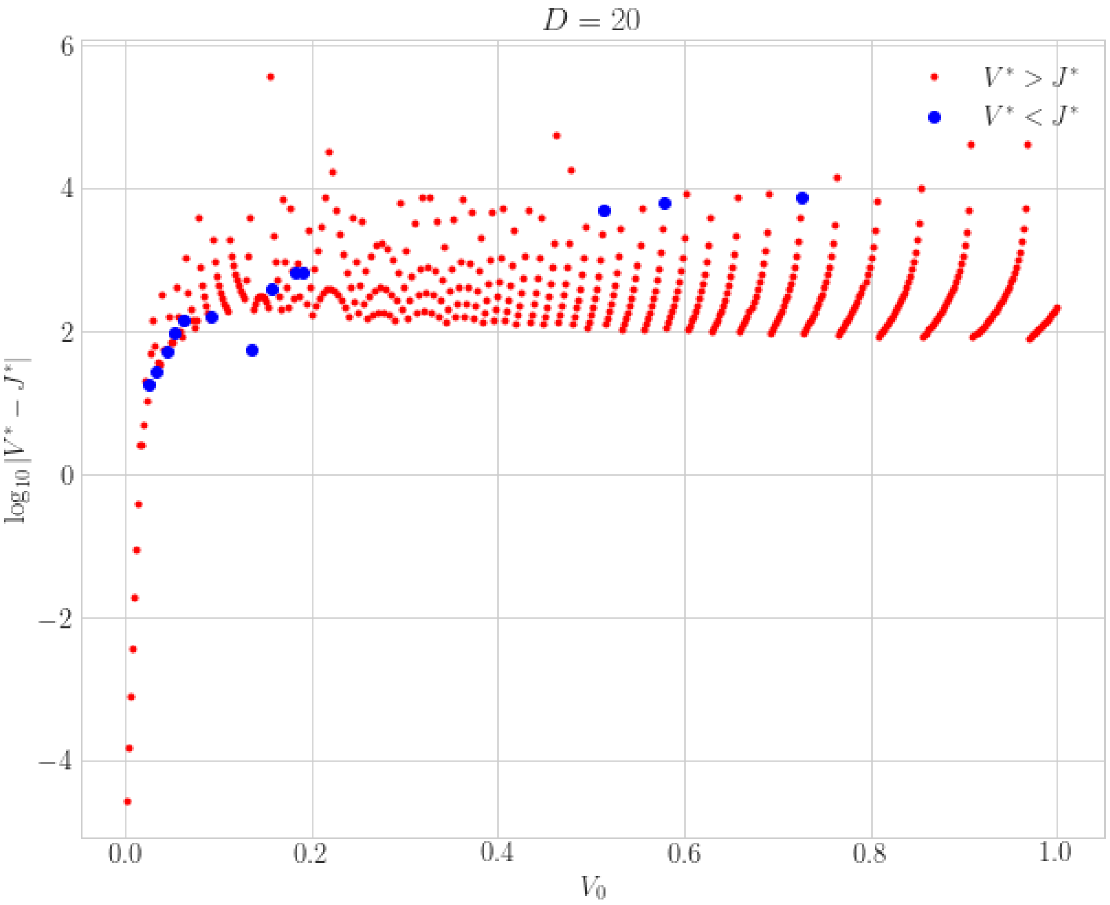

$D = 30$
● $V^* > J^*$
● $V^* < J^*$
$\log_{10}|V^* - J^*|$
4
2
0
-2
0.0
0.2
0.4
0.6
0.8
1.0
V_0

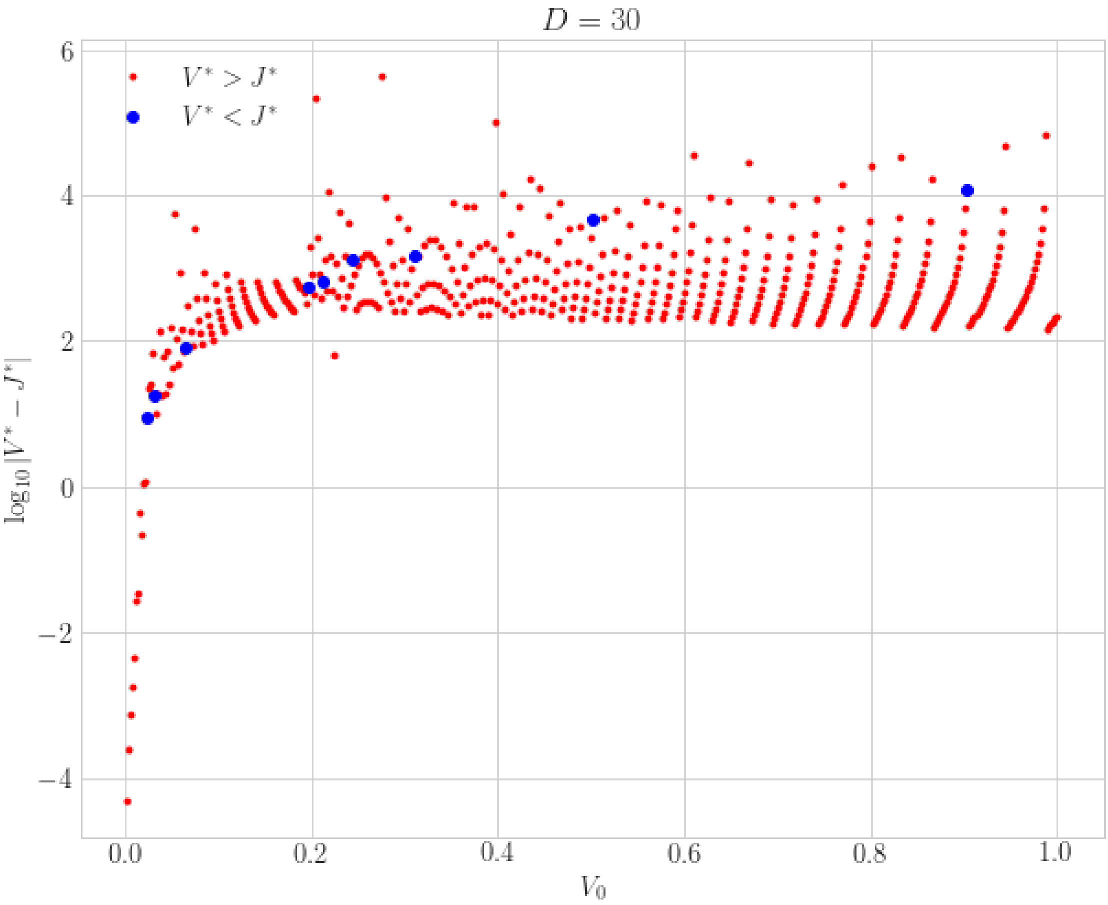

$D = 40$
● $V^* > J^*$
● $V^* < J^*$
$\log_{10}|V^* - J^*|$
4
2
0
-2
-4
0.0
0.2
0.4
0.6
0.8
1.0
V_0

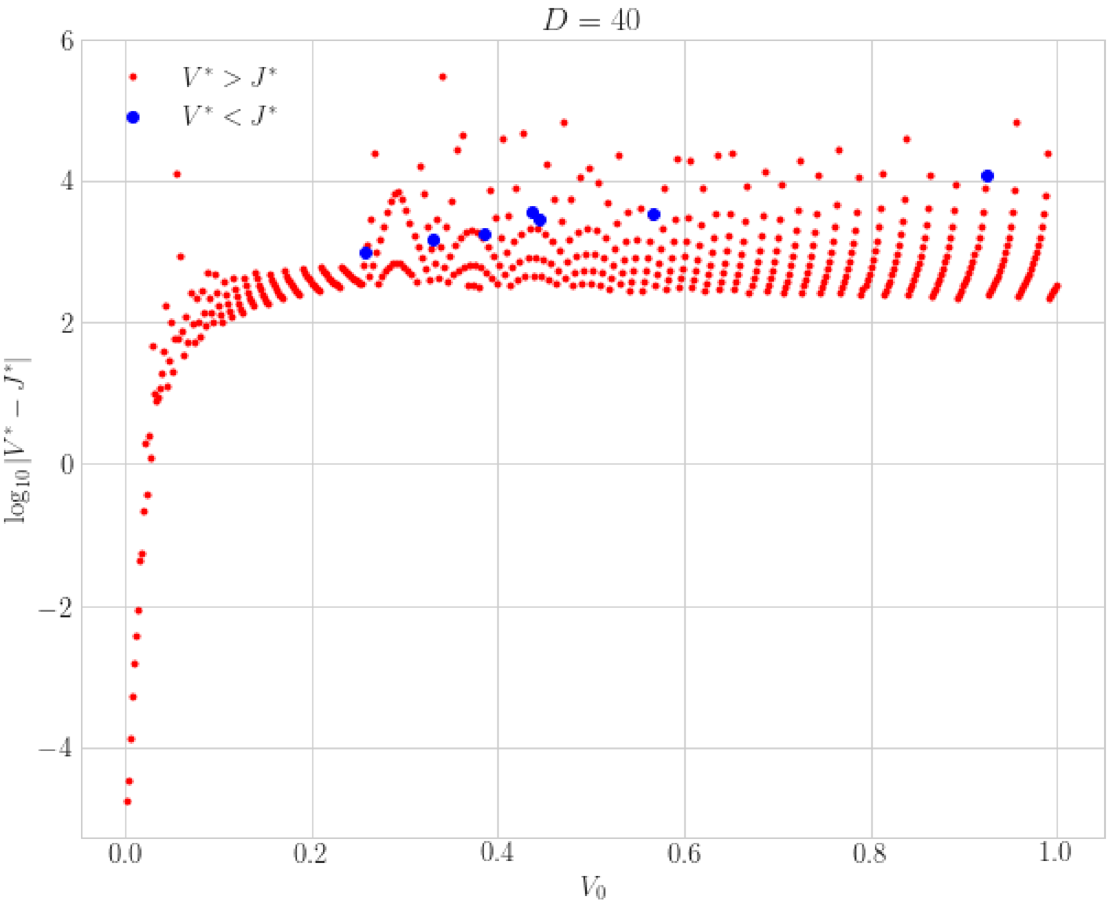


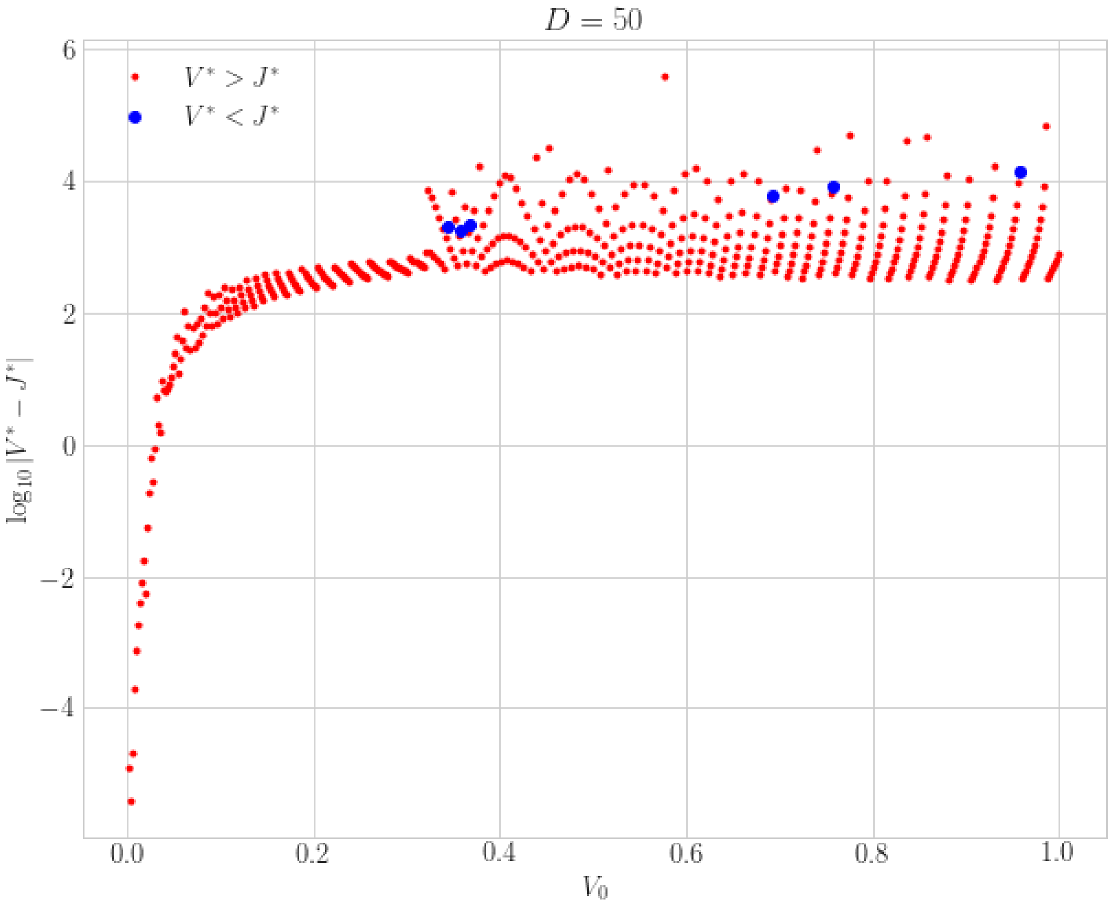
```
[24]: plot_JvsV(1, -1, r'fourth quadrant')
```


$D = 10$
$V^* > J^*$
$V^* < J^*$
$\log_{10} |V^* - J^*|$
4
2
0
-2
-4
0.0
0.2
0.4
0.6
0.8
1.0
V_0


$D = 20$
$V^* > J^*$
$V^* < J^*$
$\log_{10}|V^* - J^*|$
V_0
6
4
2
0
−2
−4
0.0
0.2
0.4
0.6
0.8
1.0


$D = 30$
● $V^* > J^*$
● $V^* < J^*$
$\log_{10}|V^* - J^*|$
6
4
2
0
−2
−4
0.0
0.2
0.4
0.6
0.8
1.0
V_0


$D = 40$
● $V^* > J^*$
● $V^* < J^*$
6
4
2
0
−2
−4
$\log_{10}|V^* - J^*|$
0.0
0.2
0.4
0.6
0.8
1.0
V_0



3.8 Behaviour of V

```
[16]: def count_Vfp(D0, V0, J0, K0, sign=1, delta=0.05):
          w_range = np.arange(-D0/2, D0/2, delta)
          U_range = np.arange(sign*delta, sign*(10 + delta), sign*delta)
          data = itertools.product(w_range, [D0], U_range, [V0], [J0], [K0])
          count = np.zeros(3)
          for outp in Pool(processes=50).starmap(complete_RG, data):
              V_fp = outp[4][-1]
              if V_fp ==0:
                  count[0] += 1
              elif V_fp > V0:
                  count[1] += 1
              elif V_fp < V0:
                  count[2] += 1
          return count

      def plot_Vcount(V0_range, count, title):
```

```
    plt.plot(V0_range, count[0], marker=".", color='r', label=r"$V^*=0$" )
    plt.plot(V0_range, count[1], marker=".", color='b', label=r"$V^* > V_0$")
    plt.plot(V0_range, count[2], marker=".", color='g', label=r"$V^* < V_0$")
    plt.legend()
    plt.title(title)
    plt.xlabel(r"$V_0$")
    plt.ylabel(r"fraction of fixed points")
    plt.show()

def plot_all(J0, K0, sign, title, V0_range=np.arange(0.001,0.101,0.001),␣
 ↪D0_range = range(10, 20, 3)):
    for D0 in D0_range:
        c0, c1, c2 = [], [], []
        for V0 in V0_range:
            print (V0)
            count = count_Vfp(D0, V0, J0, K0, sign)
            c0.append(count[0]/sum(count))
            c1.append(count[1]/sum(count))
            c2.append(count[2]/sum(count))
        plot_Vcount(V0_range, [c0, c1, c2], title+r", $D={}$".format(D0))
```

```
[17]: V0_range = np.arange(0.001,0.05,0.0002)
      plot_all(0.2, 0.1, 1, r"first quadrant", V0_range=V0_range)
```

```
0.001
0.0012000000000000001
0.0014000000000000002
0.0016000000000000003
0.0018000000000000004
0.0020000000000000005
0.0022000000000000006
0.0024000000000000007
0.0026000000000000007
0.002800000000000001
0.003000000000000001
0.003200000000000001
0.003400000000000001
0.003600000000000001
0.0038000000000000013
0.004000000000000002
0.0042000000000000015
0.004400000000000001
0.004600000000000002
0.004800000000000002
0.005000000000000002
0.005200000000000002
0.005400000000000002
0.005600000000000002
```

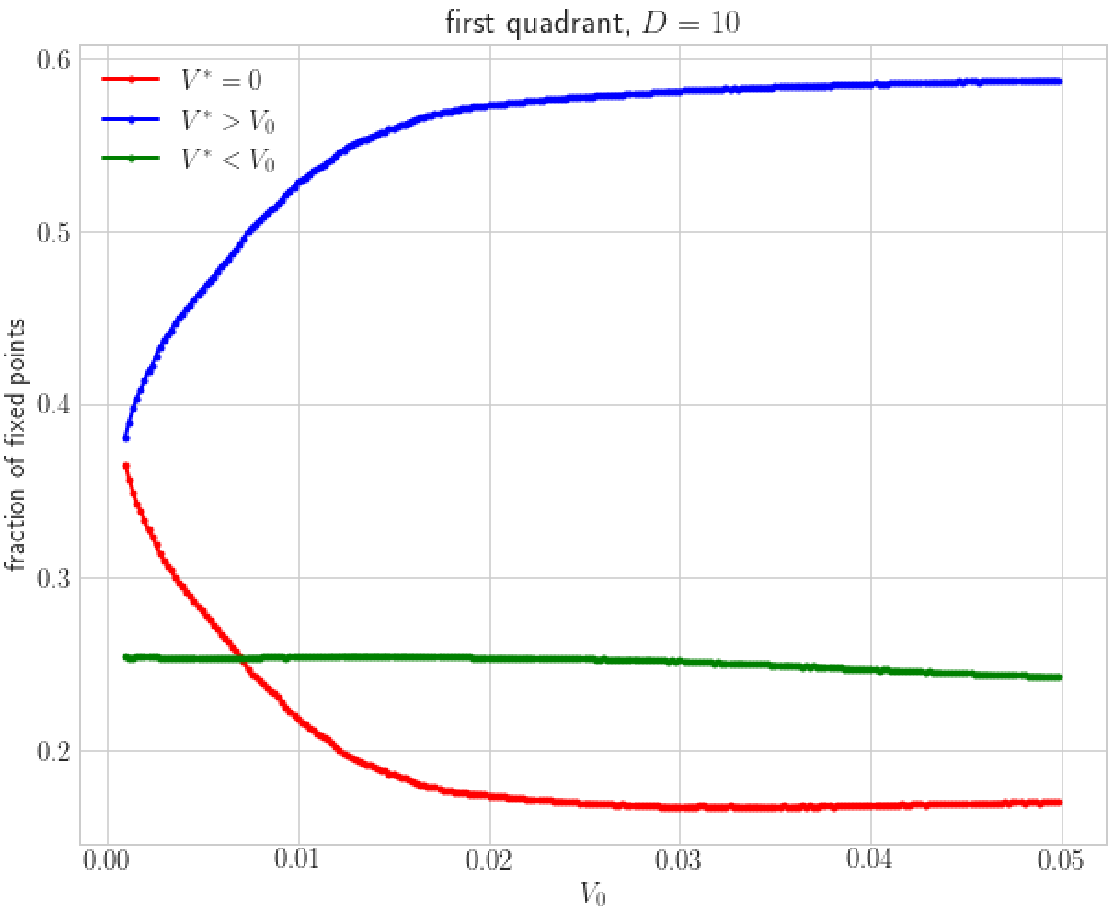
```
0.005800000000000002
0.006000000000000003
0.006200000000000002
0.006400000000000002
0.0066000000000000026
0.006800000000000003
0.007000000000000003
0.007200000000000002
0.007400000000000003
0.0076000000000000035
0.007800000000000003
0.008000000000000004
0.008200000000000002
0.008400000000000005
0.008600000000000003
0.008800000000000002
0.009000000000000005
0.009200000000000003
0.009400000000000006
0.009600000000000004
0.009800000000000003
0.010000000000000005
0.010200000000000004
0.010400000000000003
0.010600000000000005
0.010800000000000004
0.011000000000000006
0.011200000000000005
0.011400000000000004
0.011600000000000006
0.011800000000000005
0.012000000000000004
0.012200000000000006
0.012400000000000005
0.012600000000000007
0.012800000000000006
0.013000000000000005
0.013200000000000007
0.013400000000000006
0.013600000000000004
0.013800000000000007
0.014000000000000005
0.014200000000000008
0.014400000000000007
0.014600000000000005
0.014800000000000008
0.015000000000000006
0.015200000000000005
```

```
0.01540000000000007
0.01560000000000006
0.0158000000000001
0.01600000000000007
0.01620000000000006
0.0164000000000001
0.01660000000000007
0.0168000000000001
0.01700000000000008
0.01720000000000007
0.0174000000000001
0.01760000000000008
0.0178000000000001
0.0180000000000001
0.01820000000000008
0.0184000000000001
0.0186000000000001
0.01880000000000008
0.0190000000000001
0.0192000000000001
0.01940000000000008
0.0196000000000001
0.0198000000000001
0.0200000000000001
0.0202000000000001
0.0204000000000001
0.0206000000000001
0.0208000000000001
0.0210000000000001
0.0212000000000001
0.0214000000000001
0.0216000000000001
0.0218000000000001
0.0220000000000001
0.0222000000000001
0.0224000000000001
0.0226000000000001
0.0228000000000001
0.0230000000000001
0.02320000000000012
0.0234000000000001
0.0236000000000001
0.02380000000000012
0.0240000000000001
0.02420000000000013
0.02440000000000012
0.0246000000000001
0.02480000000000013
```

```
0.025000000000000012
0.02520000000000001
0.025400000000000013
0.02560000000000001
0.02580000000000001
0.026000000000000013
0.02620000000000001
0.026400000000000014
0.026600000000000013
0.02680000000000001
0.027000000000000014
0.027200000000000012
0.027400000000000015
0.027600000000000013
0.027800000000000012
0.028000000000000014
0.028200000000000013
0.028400000000000012
0.028600000000000014
0.028800000000000013
0.029000000000000012
0.029200000000000014
0.029400000000000013
0.029600000000000015
0.029800000000000014
0.030000000000000013
0.030200000000000015
0.030400000000000014
0.030600000000000016
0.030800000000000015
0.031000000000000014
0.031200000000000016
0.03140000000000001
0.03160000000000001
0.031800000000000016
0.032000000000000015
0.03220000000000001
0.03240000000000001
0.03260000000000002
0.03280000000000002
0.033000000000000015
0.033200000000000014
0.03340000000000001
0.03360000000000002
0.03380000000000002
0.034000000000000016
0.034200000000000015
0.034400000000000014
```

```
0.03460000000000002
0.03480000000000002
0.03500000000000002
0.035200000000000016
0.035400000000000015
0.035600000000000014
0.03580000000000002
0.03600000000000002
0.03620000000000002
0.036400000000000016
0.036600000000000014
0.03680000000000002
0.03700000000000002
0.03720000000000002
0.03740000000000002
0.037600000000000015
0.037800000000000014
0.03800000000000002
0.03820000000000002
0.03840000000000002
0.038600000000000016
0.038800000000000015
0.03900000000000002
0.03920000000000002
0.03940000000000002
0.03960000000000002
0.039800000000000016
0.04000000000000002
0.04020000000000002
0.04040000000000002
0.04060000000000002
0.04080000000000002
0.04100000000000002
0.04120000000000002
0.04140000000000002
0.04160000000000002
0.04180000000000002
0.042000000000000016
0.04220000000000002
0.04240000000000002
0.04260000000000002
0.04280000000000002
0.04300000000000002
0.04320000000000002
0.04340000000000002
0.04360000000000002
0.04380000000000002
0.04400000000000002
```

```
0.04420000000000002
0.04440000000000002
0.04460000000000002
0.04480000000000002
0.04500000000000002
0.04520000000000002
0.045400000000000024
0.04560000000000002
0.04580000000000002
0.04600000000000002
0.04620000000000002
0.046400000000000025
0.04660000000000002
0.04680000000000002
0.04700000000000002
0.04720000000000002
0.047400000000000025
0.047600000000000024
0.04780000000000002
0.04800000000000002
0.04820000000000002
0.04840000000000002
0.048600000000000025
0.048800000000000024
0.04900000000000002
0.04920000000000002
0.04940000000000002
0.049600000000000026
0.049800000000000025
```



```
0.001
0.0012000000000000001
0.0014000000000000002
0.0016000000000000003
0.0018000000000000004
0.0020000000000000005
0.0022000000000000006
0.0024000000000000007
0.0026000000000000007
0.002800000000000001
0.003000000000000001
0.003200000000000001
0.003400000000000001
0.003600000000000001
0.0038000000000000013
0.004000000000000002
0.0042000000000000015
0.004400000000000001
0.004600000000000002
```

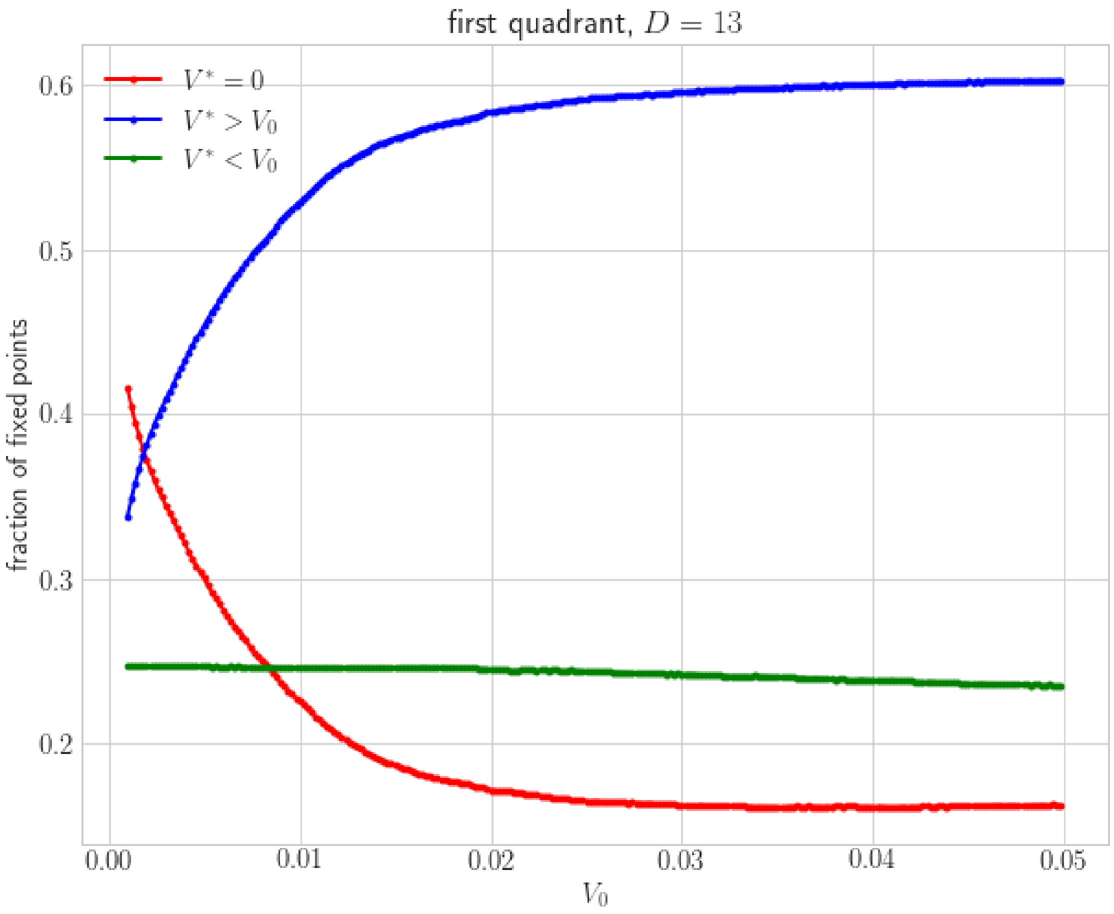
```
0.004800000000000002
0.005000000000000002
0.005200000000000002
0.005400000000000002
0.005600000000000002
0.005800000000000002
0.006000000000000003
0.006200000000000002
0.006400000000000002
0.0066000000000000026
0.006800000000000003
0.007000000000000003
0.007200000000000002
0.007400000000000003
0.0076000000000000035
0.007800000000000003
0.008000000000000004
0.008200000000000002
0.008400000000000005
0.008600000000000003
0.008800000000000002
0.009000000000000005
0.009200000000000003
0.009400000000000006
0.009600000000000004
0.009800000000000003
0.010000000000000005
0.010200000000000004
0.010400000000000003
0.010600000000000005
0.010800000000000004
0.011000000000000006
0.011200000000000005
0.011400000000000004
0.011600000000000006
0.011800000000000005
0.012000000000000004
0.012200000000000006
0.012400000000000005
0.012600000000000007
0.012800000000000006
0.013000000000000005
0.013200000000000007
0.013400000000000006
0.013600000000000004
0.013800000000000007
0.014000000000000005
0.014200000000000008
```

```
0.014400000000000007
0.014600000000000005
0.014800000000000008
0.015000000000000006
0.015200000000000005
0.015400000000000007
0.015600000000000006
0.01580000000000001
0.016000000000000007
0.016200000000000006
0.01640000000000001
0.016600000000000007
0.01680000000000001
0.017000000000000008
0.017200000000000007
0.01740000000000001
0.017600000000000008
0.01780000000000001
0.01800000000000001
0.018200000000000008
0.01840000000000001
0.01860000000000001
0.018800000000000008
0.01900000000000001
0.01920000000000001
0.019400000000000008
0.01960000000000001
0.01980000000000001
0.02000000000000001
0.02020000000000001
0.02040000000000001
0.02060000000000001
0.02080000000000001
0.02100000000000001
0.02120000000000001
0.02140000000000001
0.02160000000000001
0.02180000000000001
0.02200000000000001
0.02220000000000001
0.02240000000000001
0.02260000000000001
0.02280000000000001
0.02300000000000001
0.023200000000000012
0.02340000000000001
0.02360000000000001
0.023800000000000012
```

```
0.02400000000000001
0.024200000000000013
0.024400000000000012
0.02460000000000001
0.024800000000000013
0.025000000000000012
0.02520000000000001
0.025400000000000013
0.02560000000000001
0.02580000000000001
0.026000000000000013
0.02620000000000001
0.026400000000000014
0.026600000000000013
0.02680000000000001
0.027000000000000014
0.027200000000000012
0.027400000000000015
0.027600000000000013
0.027800000000000012
0.028000000000000014
0.028200000000000013
0.028400000000000012
0.028600000000000014
0.028800000000000013
0.029000000000000012
0.029200000000000014
0.029400000000000013
0.029600000000000015
0.029800000000000014
0.030000000000000013
0.030200000000000015
0.030400000000000014
0.030600000000000016
0.030800000000000015
0.031000000000000014
0.031200000000000016
0.03140000000000001
0.03160000000000001
0.031800000000000016
0.032000000000000015
0.03220000000000001
0.03240000000000001
0.03260000000000002
0.03280000000000002
0.033000000000000015
0.033200000000000014
0.03340000000000001
```

```
0.03360000000000002
0.03380000000000002
0.034000000000000016
0.034200000000000015
0.034400000000000014
0.03460000000000002
0.03480000000000002
0.03500000000000002
0.035200000000000016
0.035400000000000015
0.035600000000000014
0.03580000000000002
0.03600000000000002
0.03620000000000002
0.036400000000000016
0.036600000000000014
0.03680000000000002
0.03700000000000002
0.03720000000000002
0.03740000000000002
0.037600000000000015
0.037800000000000014
0.03800000000000002
0.03820000000000002
0.03840000000000002
0.038600000000000016
0.038800000000000015
0.03900000000000002
0.03920000000000002
0.03940000000000002
0.03960000000000002
0.039800000000000016
0.04000000000000002
0.04020000000000002
0.04040000000000002
0.04060000000000002
0.04080000000000002
0.04100000000000002
0.04120000000000002
0.04140000000000002
0.04160000000000002
0.04180000000000002
0.042000000000000016
0.04220000000000002
0.04240000000000002
0.04260000000000002
0.04280000000000002
0.04300000000000002
```

```
0.04320000000000002
0.04340000000000002
0.04360000000000002
0.04380000000000002
0.04400000000000002
0.04420000000000002
0.04440000000000002
0.04460000000000002
0.04480000000000002
0.04500000000000002
0.04520000000000002
0.045400000000000024
0.04560000000000002
0.04580000000000002
0.04600000000000002
0.04620000000000002
0.046400000000000025
0.04660000000000002
0.04680000000000002
0.04700000000000002
0.04720000000000002
0.047400000000000025
0.047600000000000024
0.04780000000000002
0.04800000000000002
0.04820000000000002
0.04840000000000002
0.048600000000000025
0.048800000000000024
0.04900000000000002
0.04920000000000002
0.04940000000000002
0.049600000000000026
0.049800000000000025
```



```
0.001
0.0012000000000000001
0.0014000000000000002
0.0016000000000000003
0.0018000000000000004
0.0020000000000000005
0.0022000000000000006
0.0024000000000000007
0.0026000000000000007
0.002800000000000001
0.003000000000000001
0.003200000000000001
0.003400000000000001
0.003600000000000001
0.0038000000000000013
0.004000000000000002
0.0042000000000000015
0.004400000000000001
0.004600000000000002
```

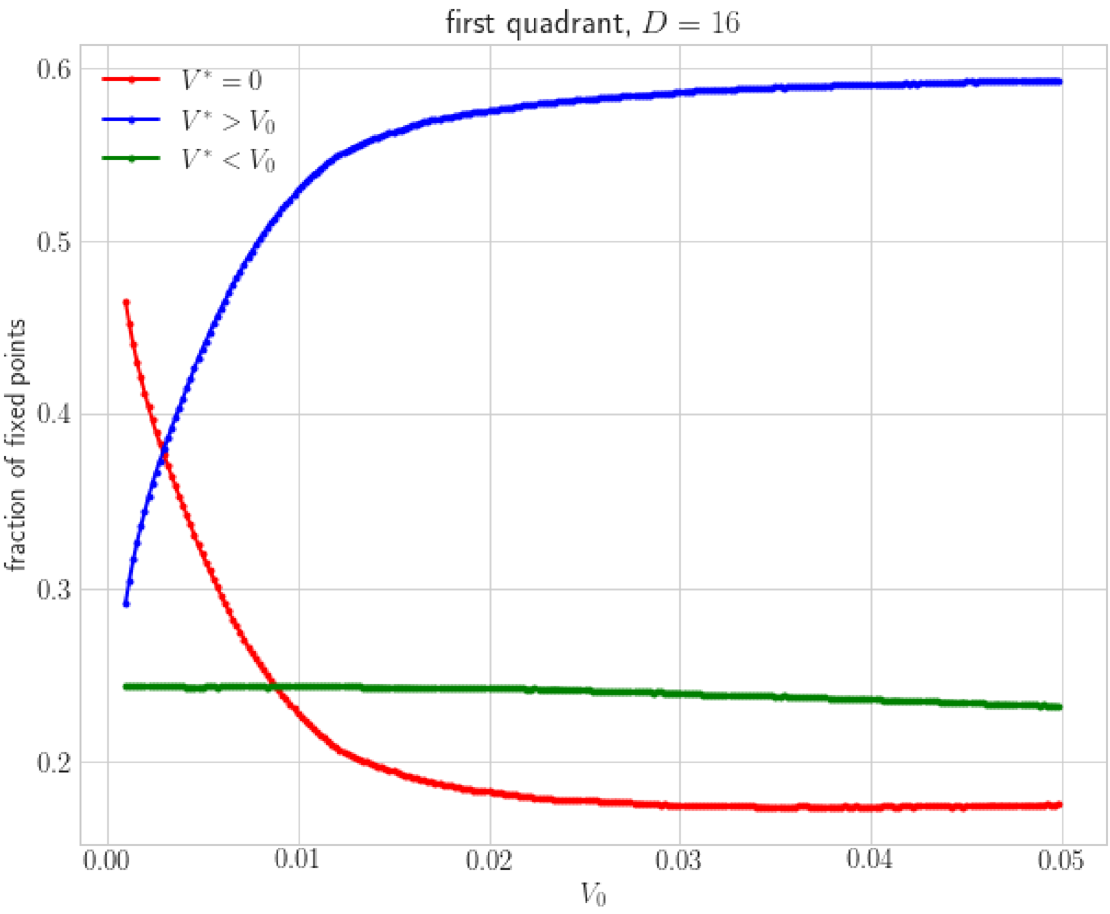
```
0.004800000000000002
0.005000000000000002
0.005200000000000002
0.005400000000000002
0.005600000000000002
0.005800000000000002
0.006000000000000003
0.006200000000000002
0.006400000000000002
0.0066000000000000026
0.006800000000000003
0.007000000000000003
0.007200000000000002
0.007400000000000003
0.0076000000000000035
0.007800000000000003
0.008000000000000004
0.008200000000000002
0.008400000000000005
0.008600000000000003
0.008800000000000002
0.009000000000000005
0.009200000000000003
0.009400000000000006
0.009600000000000004
0.009800000000000003
0.010000000000000005
0.010200000000000004
0.010400000000000003
0.010600000000000005
0.010800000000000004
0.011000000000000006
0.011200000000000005
0.011400000000000004
0.011600000000000006
0.011800000000000005
0.012000000000000004
0.012200000000000006
0.012400000000000005
0.012600000000000007
0.012800000000000006
0.013000000000000005
0.013200000000000007
0.013400000000000006
0.013600000000000004
0.013800000000000007
0.014000000000000005
0.014200000000000008
```

```
0.014400000000000007
0.014600000000000005
0.014800000000000008
0.015000000000000006
0.015200000000000005
0.015400000000000007
0.015600000000000006
0.01580000000000001
0.016000000000000007
0.016200000000000006
0.01640000000000001
0.016600000000000007
0.01680000000000001
0.017000000000000008
0.017200000000000007
0.01740000000000001
0.017600000000000008
0.01780000000000001
0.01800000000000001
0.018200000000000008
0.01840000000000001
0.01860000000000001
0.018800000000000008
0.01900000000000001
0.01920000000000001
0.019400000000000008
0.01960000000000001
0.01980000000000001
0.02000000000000001
0.02020000000000001
0.02040000000000001
0.02060000000000001
0.02080000000000001
0.02100000000000001
0.02120000000000001
0.02140000000000001
0.02160000000000001
0.02180000000000001
0.02200000000000001
0.02220000000000001
0.02240000000000001
0.02260000000000001
0.02280000000000001
0.02300000000000001
0.023200000000000012
0.02340000000000001
0.02360000000000001
0.023800000000000012
```

```
0.02400000000000001
0.024200000000000013
0.024400000000000012
0.02460000000000001
0.024800000000000013
0.025000000000000012
0.02520000000000001
0.025400000000000013
0.02560000000000001
0.02580000000000001
0.026000000000000013
0.02620000000000001
0.026400000000000014
0.026600000000000013
0.02680000000000001
0.027000000000000014
0.027200000000000012
0.027400000000000015
0.027600000000000013
0.027800000000000012
0.028000000000000014
0.028200000000000013
0.028400000000000012
0.028600000000000014
0.028800000000000013
0.029000000000000012
0.029200000000000014
0.029400000000000013
0.029600000000000015
0.029800000000000014
0.030000000000000013
0.030200000000000015
0.030400000000000014
0.030600000000000016
0.030800000000000015
0.031000000000000014
0.031200000000000016
0.03140000000000001
0.03160000000000001
0.031800000000000016
0.032000000000000015
0.03220000000000001
0.03240000000000001
0.03260000000000002
0.03280000000000002
0.033000000000000015
0.033200000000000014
0.03340000000000001
```

```
0.03360000000000002
0.03380000000000002
0.034000000000000016
0.034200000000000015
0.034400000000000014
0.03460000000000002
0.03480000000000002
0.03500000000000002
0.035200000000000016
0.035400000000000015
0.035600000000000014
0.03580000000000002
0.03600000000000002
0.03620000000000002
0.036400000000000016
0.036600000000000014
0.03680000000000002
0.03700000000000002
0.03720000000000002
0.03740000000000002
0.037600000000000015
0.037800000000000014
0.03800000000000002
0.03820000000000002
0.03840000000000002
0.038600000000000016
0.038800000000000015
0.03900000000000002
0.03920000000000002
0.03940000000000002
0.03960000000000002
0.039800000000000016
0.04000000000000002
0.04020000000000002
0.04040000000000002
0.04060000000000002
0.04080000000000002
0.04100000000000002
0.04120000000000002
0.04140000000000002
0.04160000000000002
0.04180000000000002
0.042000000000000016
0.04220000000000002
0.04240000000000002
0.04260000000000002
0.04280000000000002
0.04300000000000002
```

```
0.04320000000000002
0.04340000000000002
0.04360000000000002
0.04380000000000002
0.04400000000000002
0.04420000000000002
0.04440000000000002
0.04460000000000002
0.04480000000000002
0.04500000000000002
0.04520000000000002
0.045400000000000024
0.04560000000000002
0.04580000000000002
0.04600000000000002
0.04620000000000002
0.046400000000000025
0.04660000000000002
0.04680000000000002
0.04700000000000002
0.04720000000000002
0.047400000000000025
0.047600000000000024
0.04780000000000002
0.04800000000000002
0.04820000000000002
0.04840000000000002
0.048600000000000025
0.048800000000000024
0.04900000000000002
0.04920000000000002
0.04940000000000002
0.049600000000000026
0.049800000000000025
```



```
0.001
0.0012000000000000001
0.0014000000000000002
0.0016000000000000003
0.0018000000000000004
0.0020000000000000005
0.0022000000000000006
0.0024000000000000007
0.0026000000000000007
0.002800000000000001
0.003000000000000001
0.003200000000000001
0.003400000000000001
0.003600000000000001
0.0038000000000000013
0.004000000000000002
0.0042000000000000015
0.004400000000000001
0.004600000000000002
```

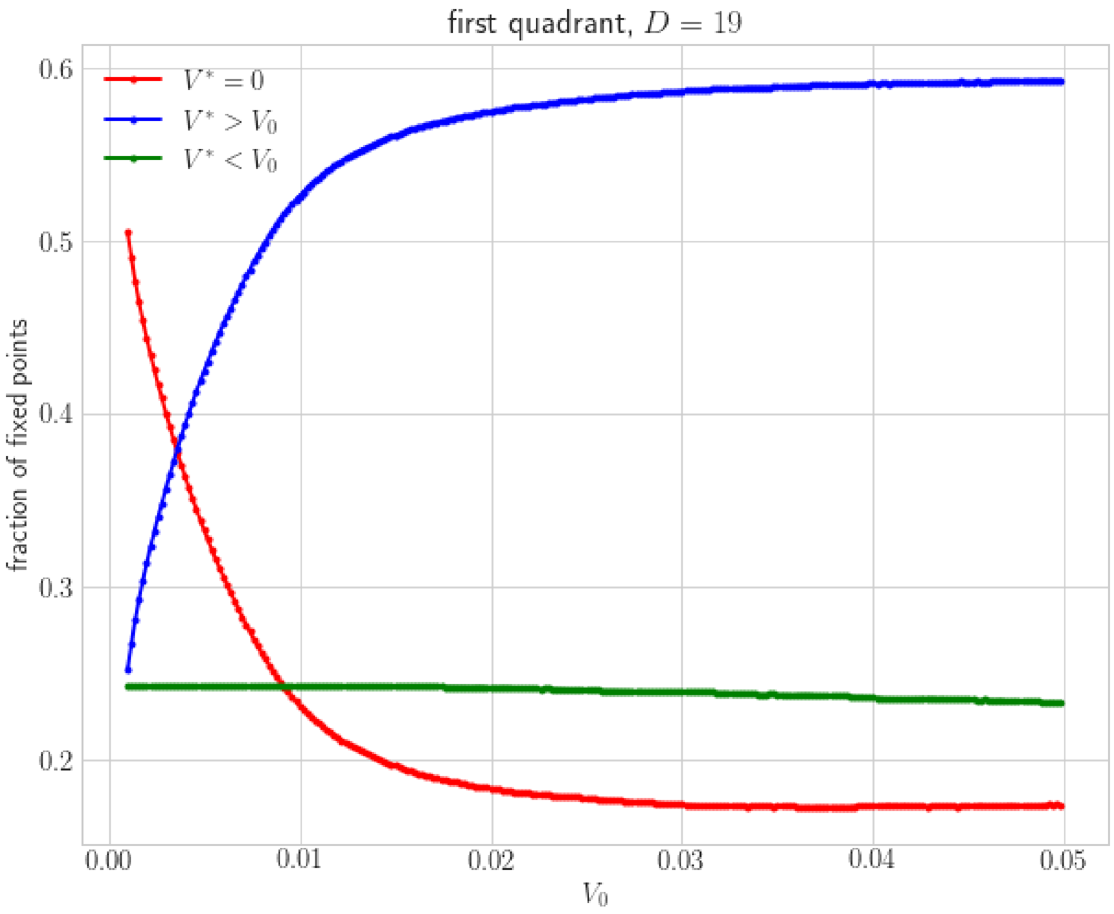
```
0.004800000000000002
0.005000000000000002
0.005200000000000002
0.005400000000000002
0.005600000000000002
0.005800000000000002
0.006000000000000003
0.006200000000000002
0.006400000000000002
0.0066000000000000026
0.006800000000000003
0.007000000000000003
0.007200000000000002
0.007400000000000003
0.0076000000000000035
0.007800000000000003
0.008000000000000004
0.008200000000000002
0.008400000000000005
0.008600000000000003
0.008800000000000002
0.009000000000000005
0.009200000000000003
0.009400000000000006
0.009600000000000004
0.009800000000000003
0.010000000000000005
0.010200000000000004
0.010400000000000003
0.010600000000000005
0.010800000000000004
0.011000000000000006
0.011200000000000005
0.011400000000000004
0.011600000000000006
0.011800000000000005
0.012000000000000004
0.012200000000000006
0.012400000000000005
0.012600000000000007
0.012800000000000006
0.013000000000000005
0.013200000000000007
0.013400000000000006
0.013600000000000004
0.013800000000000007
0.014000000000000005
0.014200000000000008
```

```
0.014400000000000007
0.014600000000000005
0.014800000000000008
0.015000000000000006
0.015200000000000005
0.015400000000000007
0.015600000000000006
0.01580000000000001
0.016000000000000007
0.016200000000000006
0.01640000000000001
0.016600000000000007
0.01680000000000001
0.017000000000000008
0.017200000000000007
0.01740000000000001
0.017600000000000008
0.01780000000000001
0.01800000000000001
0.018200000000000008
0.01840000000000001
0.01860000000000001
0.018800000000000008
0.01900000000000001
0.01920000000000001
0.019400000000000008
0.01960000000000001
0.01980000000000001
0.02000000000000001
0.02020000000000001
0.02040000000000001
0.02060000000000001
0.02080000000000001
0.02100000000000001
0.02120000000000001
0.02140000000000001
0.02160000000000001
0.02180000000000001
0.02200000000000001
0.02220000000000001
0.02240000000000001
0.02260000000000001
0.02280000000000001
0.02300000000000001
0.023200000000000012
0.02340000000000001
0.02360000000000001
0.023800000000000012
```

```
0.02400000000000001
0.024200000000000013
0.024400000000000012
0.02460000000000001
0.024800000000000013
0.025000000000000012
0.02520000000000001
0.025400000000000013
0.02560000000000001
0.02580000000000001
0.026000000000000013
0.02620000000000001
0.026400000000000014
0.026600000000000013
0.02680000000000001
0.027000000000000014
0.027200000000000012
0.027400000000000015
0.027600000000000013
0.027800000000000012
0.028000000000000014
0.028200000000000013
0.028400000000000012
0.028600000000000014
0.028800000000000013
0.029000000000000012
0.029200000000000014
0.029400000000000013
0.029600000000000015
0.029800000000000014
0.030000000000000013
0.030200000000000015
0.030400000000000014
0.030600000000000016
0.030800000000000015
0.031000000000000014
0.031200000000000016
0.03140000000000001
0.03160000000000001
0.031800000000000016
0.032000000000000015
0.03220000000000001
0.03240000000000001
0.03260000000000002
0.03280000000000002
0.033000000000000015
0.033200000000000014
0.03340000000000001
```

```
0.03360000000000002
0.03380000000000002
0.034000000000000016
0.034200000000000015
0.034400000000000014
0.03460000000000002
0.03480000000000002
0.03500000000000002
0.035200000000000016
0.035400000000000015
0.035600000000000014
0.03580000000000002
0.03600000000000002
0.03620000000000002
0.036400000000000016
0.036600000000000014
0.03680000000000002
0.03700000000000002
0.03720000000000002
0.03740000000000002
0.037600000000000015
0.037800000000000014
0.03800000000000002
0.03820000000000002
0.03840000000000002
0.038600000000000016
0.038800000000000015
0.03900000000000002
0.03920000000000002
0.03940000000000002
0.03960000000000002
0.039800000000000016
0.04000000000000002
0.04020000000000002
0.04040000000000002
0.04060000000000002
0.04080000000000002
0.04100000000000002
0.04120000000000002
0.04140000000000002
0.04160000000000002
0.04180000000000002
0.042000000000000016
0.04220000000000002
0.04240000000000002
0.04260000000000002
0.04280000000000002
0.04300000000000002
```

```
0.04320000000000002
0.04340000000000002
0.04360000000000002
0.04380000000000002
0.04400000000000002
0.04420000000000002
0.04440000000000002
0.04460000000000002
0.04480000000000002
0.04500000000000002
0.04520000000000002
0.045400000000000024
0.04560000000000002
0.04580000000000002
0.04600000000000002
0.04620000000000002
0.046400000000000025
0.04660000000000002
0.04680000000000002
0.04700000000000002
0.04720000000000002
0.047400000000000025
0.047600000000000024
0.04780000000000002
0.04800000000000002
0.04820000000000002
0.04840000000000002
0.048600000000000025
0.048800000000000024
0.04900000000000002
0.04920000000000002
0.04940000000000002
0.049600000000000026
0.049800000000000025
```

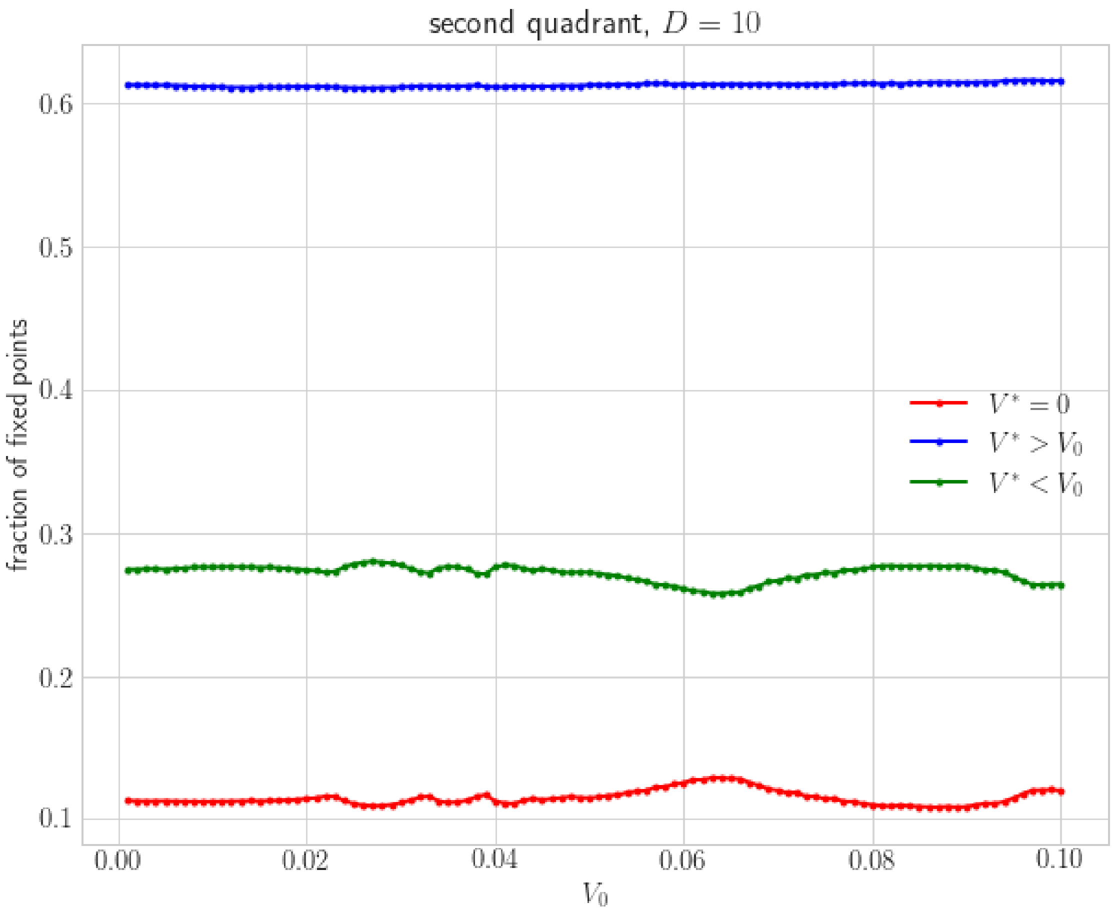


[18]:
```
plot_all(0.1, 0.2, 1, r"second quadrant")
```

```
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009000000000000001
0.010000000000000002
0.011
0.012
0.013000000000000001
0.014000000000000002
0.015
0.016
0.017
```

```
0.018000000000000002
0.019000000000000003
0.02
0.021
0.022000000000000002
0.023
0.024
0.025
0.026000000000000002
0.027000000000000003
0.028
0.029
0.030000000000000002
0.031
0.032
0.033
0.034
0.035
0.036000000000000004
0.037000000000000005
0.038
0.039
0.04
0.041
0.042
0.043000000000000003
0.044000000000000004
0.045
0.046
0.047
0.048
0.049
0.05
0.051000000000000004
0.052000000000000005
0.053000000000000005
0.054
0.055
0.056
0.057
0.058
0.059000000000000004
0.060000000000000005
0.061
0.062
0.063
0.064
0.065
```

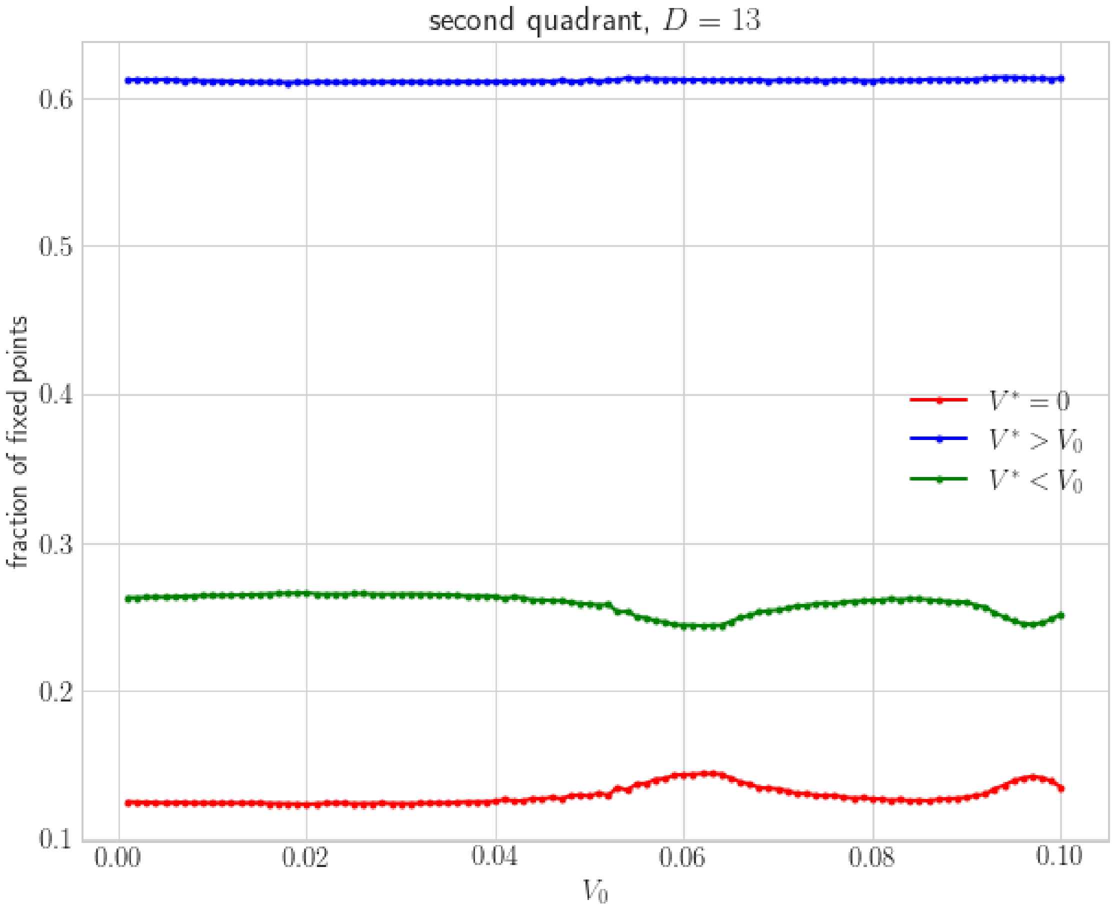
```
0.066
0.067
0.068
0.069
0.07
0.07100000000000001
0.07200000000000001
0.07300000000000001
0.074
0.075
0.076
0.077
0.078
0.079
0.08
0.081
0.082
0.083
0.084
0.085
0.08600000000000001
0.08700000000000001
0.08800000000000001
0.089
0.09
0.091
0.092
0.093
0.094
0.095
0.096
0.097
0.098
0.099
0.1
```



```
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009000000000000001
0.010000000000000002
0.011
0.012
0.013000000000000001
0.014000000000000002
0.015
0.016
0.017
0.018000000000000002
0.019000000000000003
```

```
0.02
0.021
0.022000000000000002
0.023
0.024
0.025
0.026000000000000002
0.027000000000000003
0.028
0.029
0.030000000000000002
0.031
0.032
0.033
0.034
0.035
0.036000000000000004
0.037000000000000005
0.038
0.039
0.04
0.041
0.042
0.043000000000000003
0.044000000000000004
0.045
0.046
0.047
0.048
0.049
0.05
0.051000000000000004
0.052000000000000005
0.053000000000000005
0.054
0.055
0.056
0.057
0.058
0.059000000000000004
0.060000000000000005
0.061
0.062
0.063
0.064
0.065
0.066
0.067
```

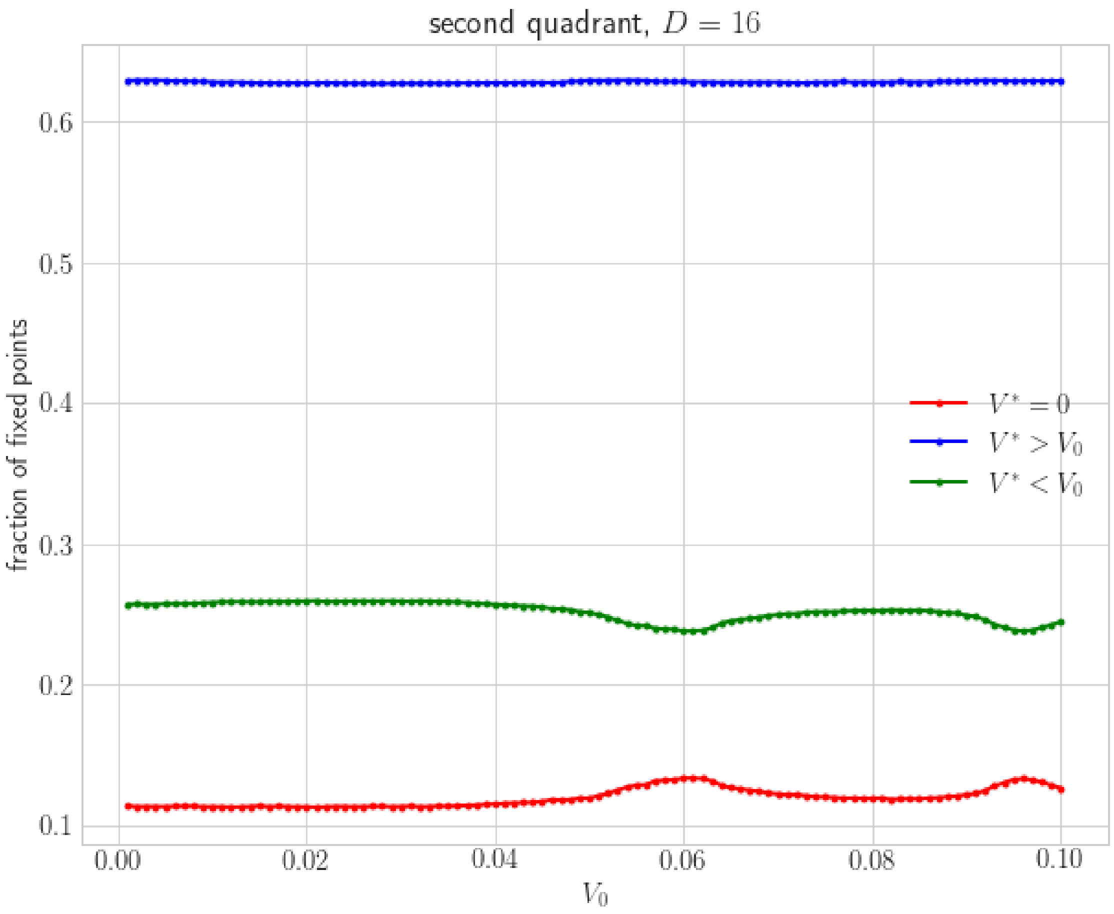
```
0.068
0.069
0.07
0.07100000000000001
0.07200000000000001
0.07300000000000001
0.074
0.075
0.076
0.077
0.078
0.079
0.08
0.081
0.082
0.083
0.084
0.085
0.08600000000000001
0.08700000000000001
0.08800000000000001
0.089
0.09
0.091
0.092
0.093
0.094
0.095
0.096
0.097
0.098
0.099
0.1
```



```
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009000000000000001
0.010000000000000002
0.011
0.012
0.013000000000000001
0.014000000000000002
0.015
0.016
0.017
0.018000000000000002
0.019000000000000003
```

```
0.02
0.021
0.022000000000000002
0.023
0.024
0.025
0.026000000000000002
0.027000000000000003
0.028
0.029
0.030000000000000002
0.031
0.032
0.033
0.034
0.035
0.036000000000000004
0.037000000000000005
0.038
0.039
0.04
0.041
0.042
0.043000000000000003
0.044000000000000004
0.045
0.046
0.047
0.048
0.049
0.05
0.051000000000000004
0.052000000000000005
0.053000000000000005
0.054
0.055
0.056
0.057
0.058
0.059000000000000004
0.060000000000000005
0.061
0.062
0.063
0.064
0.065
0.066
0.067
```

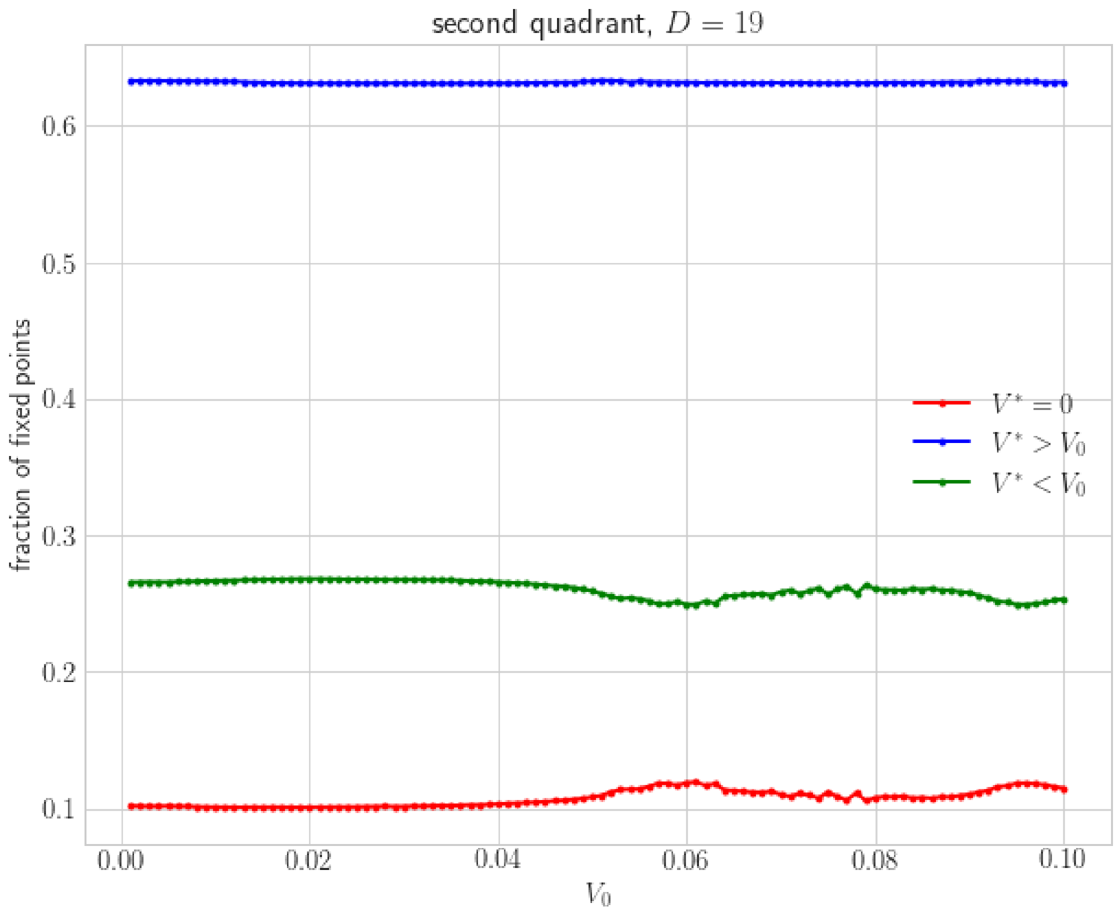
```
0.068
0.069
0.07
0.07100000000000001
0.07200000000000001
0.07300000000000001
0.074
0.075
0.076
0.077
0.078
0.079
0.08
0.081
0.082
0.083
0.084
0.085
0.08600000000000001
0.08700000000000001
0.08800000000000001
0.089
0.09
0.091
0.092
0.093
0.094
0.095
0.096
0.097
0.098
0.099
0.1
```



```
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009000000000000001
0.010000000000000002
0.011
0.012
0.013000000000000001
0.014000000000000002
0.015
0.016
0.017
0.018000000000000002
0.019000000000000003
```

```
0.02
0.021
0.022000000000000002
0.023
0.024
0.025
0.026000000000000002
0.027000000000000003
0.028
0.029
0.030000000000000002
0.031
0.032
0.033
0.034
0.035
0.036000000000000004
0.037000000000000005
0.038
0.039
0.04
0.041
0.042
0.043000000000000003
0.044000000000000004
0.045
0.046
0.047
0.048
0.049
0.05
0.051000000000000004
0.052000000000000005
0.053000000000000005
0.054
0.055
0.056
0.057
0.058
0.059000000000000004
0.060000000000000005
0.061
0.062
0.063
0.064
0.065
0.066
0.067
```

```
0.068
0.069
0.07
0.07100000000000001
0.07200000000000001
0.07300000000000001
0.074
0.075
0.076
0.077
0.078
0.079
0.08
0.081
0.082
0.083
0.084
0.085
0.08600000000000001
0.08700000000000001
0.08800000000000001
0.089
0.09
0.091
0.092
0.093
0.094
0.095
0.096
0.097
0.098
0.099
0.1
```

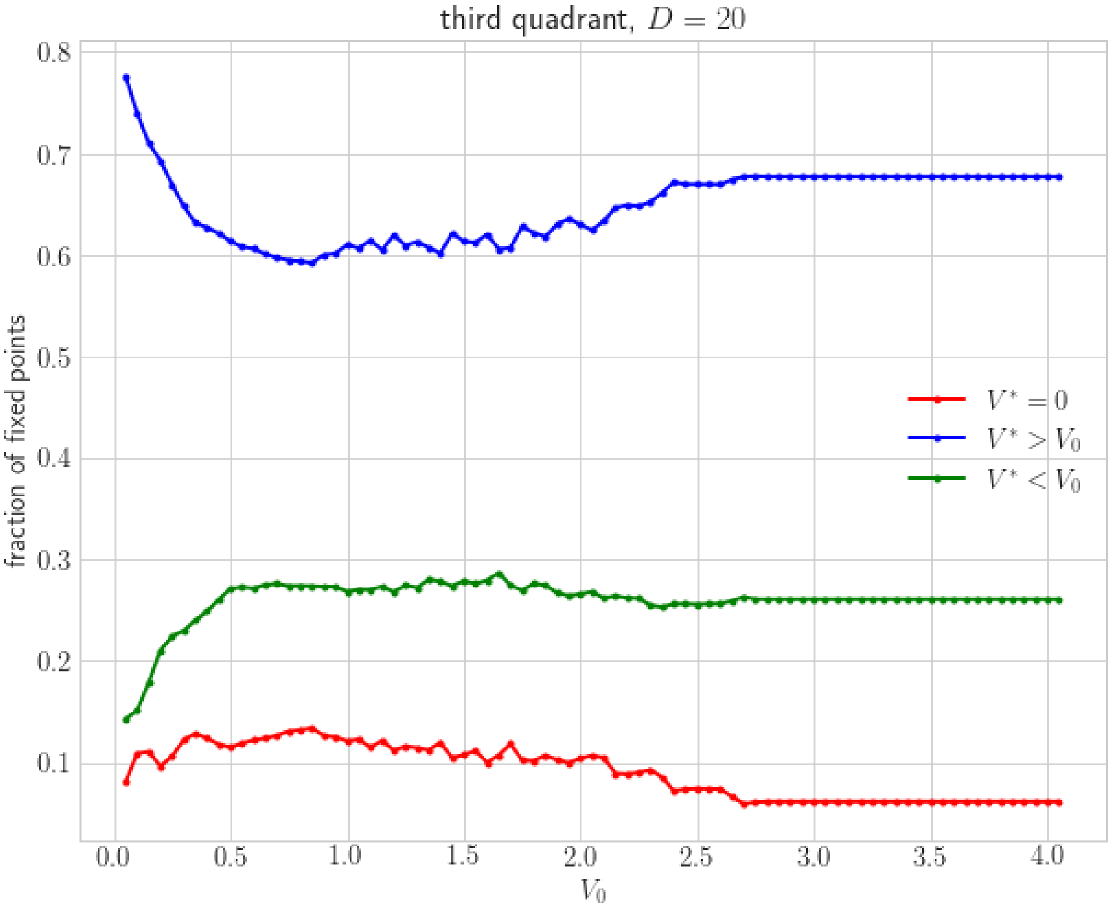


```
[19]: plot_all(0.1, 0.2, -1, r"third quadrant", V0_range=np.arange(0.05,4.1,0.05),␣
       ↪D0_range=np.arange(20, 41, 5))
```

```
0.05
0.1
0.15000000000000002
0.2
0.25
0.3
0.35000000000000003
0.4
0.45
0.5
0.55
0.6000000000000001
0.6500000000000001
0.7000000000000001
0.7500000000000001
0.8
```

```
0.8500000000000001
0.9000000000000001
0.9500000000000001
1.0
1.05
1.1
1.1500000000000001
1.2000000000000002
1.2500000000000002
1.3
1.35
1.4000000000000001
1.4500000000000002
1.5000000000000002
1.55
1.6
1.6500000000000001
1.7000000000000002
1.7500000000000002
1.8
1.85
1.9000000000000001
1.9500000000000002
2.0
2.05
2.1
2.15
2.1999999999999997
2.25
2.3
2.35
2.4
2.45
2.5
2.55
2.6
2.65
2.7
2.75
2.8
2.85
2.9
2.95
3.0
3.05
3.1
3.15
3.2
```

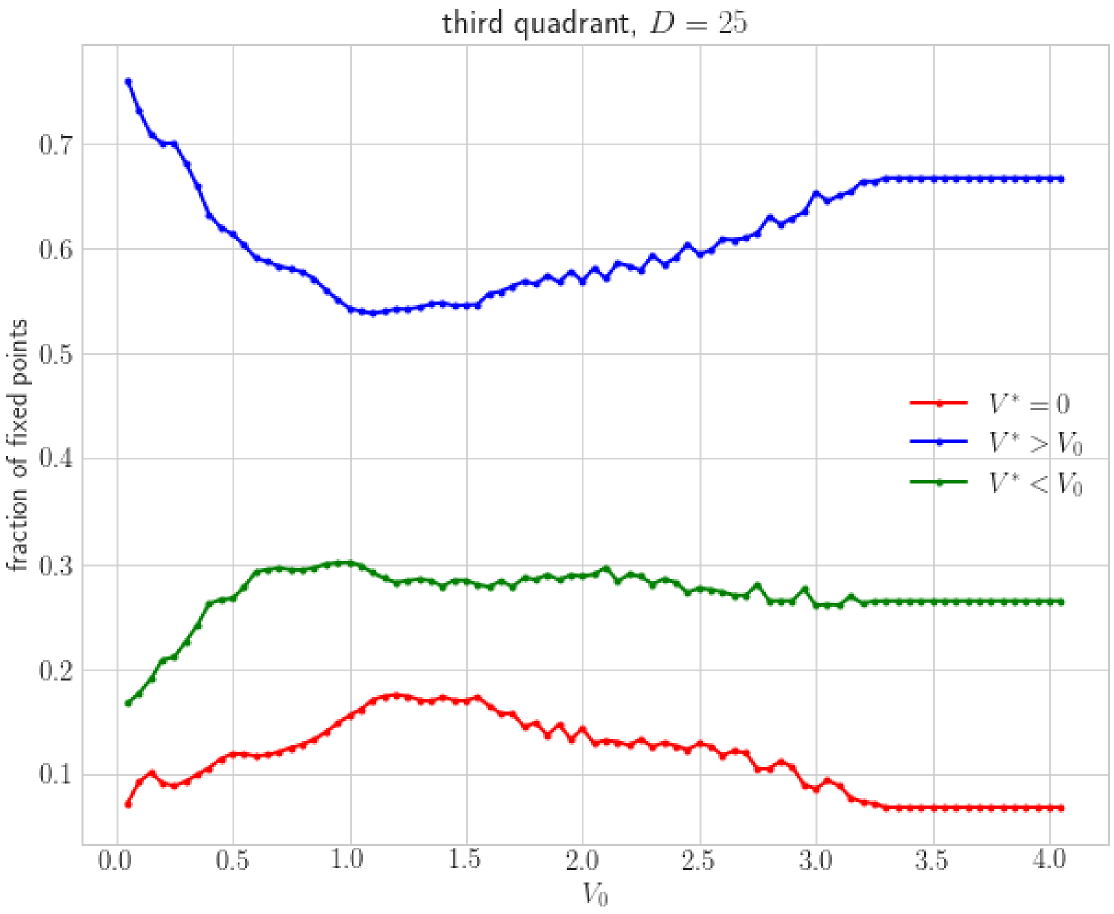
```
3.25
3.3
3.35
3.4
3.45
3.5
3.55
3.6
3.65
3.7
3.75
3.8
3.85
3.9
3.95
4.0
4.05
```



```
0.05
```

```
0.1
0.15000000000000002
0.2
0.25
0.3
0.35000000000000003
0.4
0.45
0.5
0.55
0.6000000000000001
0.6500000000000001
0.7000000000000001
0.7500000000000001
0.8
0.8500000000000001
0.9000000000000001
0.9500000000000001
1.0
1.05
1.1
1.1500000000000001
1.2000000000000002
1.2500000000000002
1.3
1.35
1.4000000000000001
1.4500000000000002
1.5000000000000002
1.55
1.6
1.6500000000000001
1.7000000000000002
1.7500000000000002
1.8
1.85
1.9000000000000001
1.9500000000000002
2.0
2.05
2.1
2.15
2.1999999999999997
2.25
2.3
2.35
2.4
2.45
```

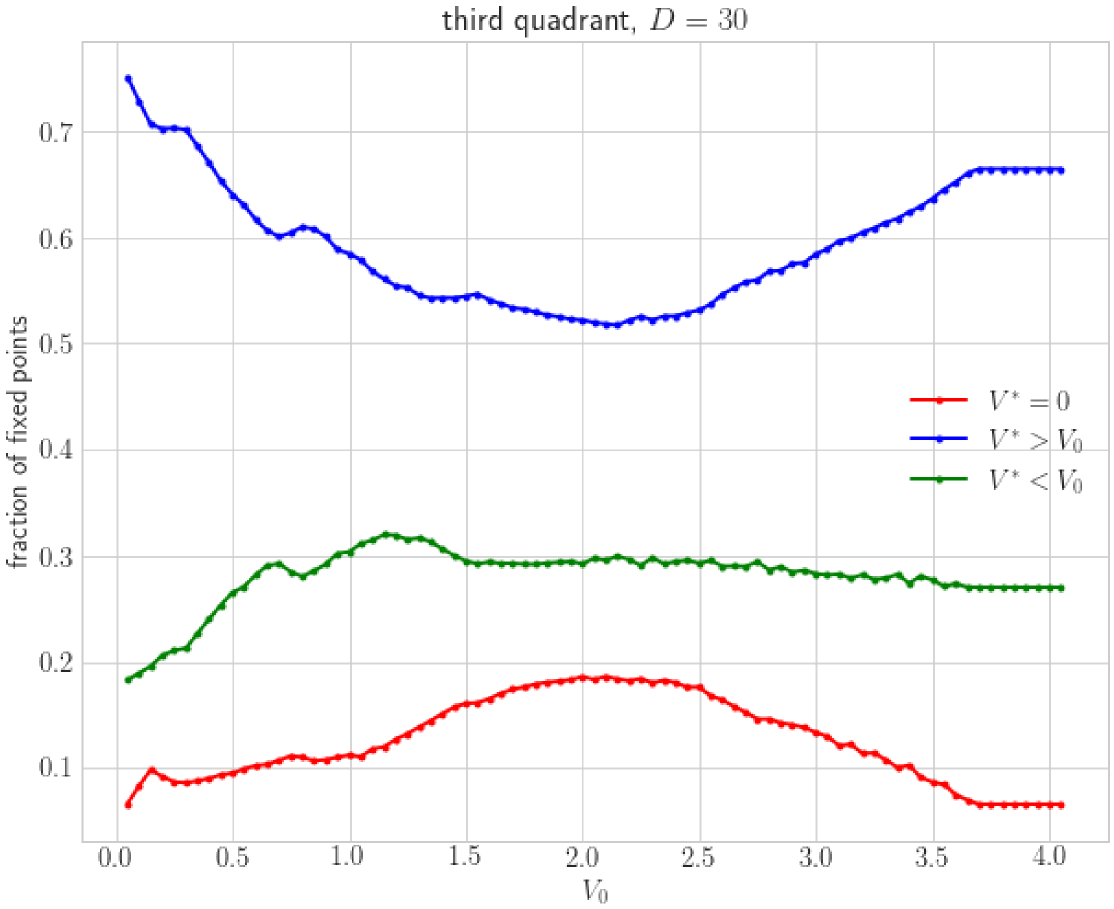
```
2.5
2.55
2.6
2.65
2.7
2.75
2.8
2.85
2.9
2.95
3.0
3.05
3.1
3.15
3.2
3.25
3.3
3.35
3.4
3.45
3.5
3.55
3.6
3.65
3.7
3.75
3.8
3.85
3.9
3.95
4.0
4.05
```



```
0.05
0.1
0.15000000000000002
0.2
0.25
0.3
0.35000000000000003
0.4
0.45
0.5
0.55
0.6000000000000001
0.6500000000000001
0.7000000000000001
0.7500000000000001
0.8
0.8500000000000001
0.9000000000000001
0.9500000000000001
```

```
1.0
1.05
1.1
1.1500000000000001
1.2000000000000002
1.2500000000000002
1.3
1.35
1.4000000000000001
1.4500000000000002
1.5000000000000002
1.55
1.6
1.6500000000000001
1.7000000000000002
1.7500000000000002
1.8
1.85
1.9000000000000001
1.9500000000000002
2.0
2.05
2.1
2.15
2.1999999999999997
2.25
2.3
2.35
2.4
2.45
2.5
2.55
2.6
2.65
2.7
2.75
2.8
2.85
2.9
2.95
3.0
3.05
3.1
3.15
3.2
3.25
3.3
3.35
```

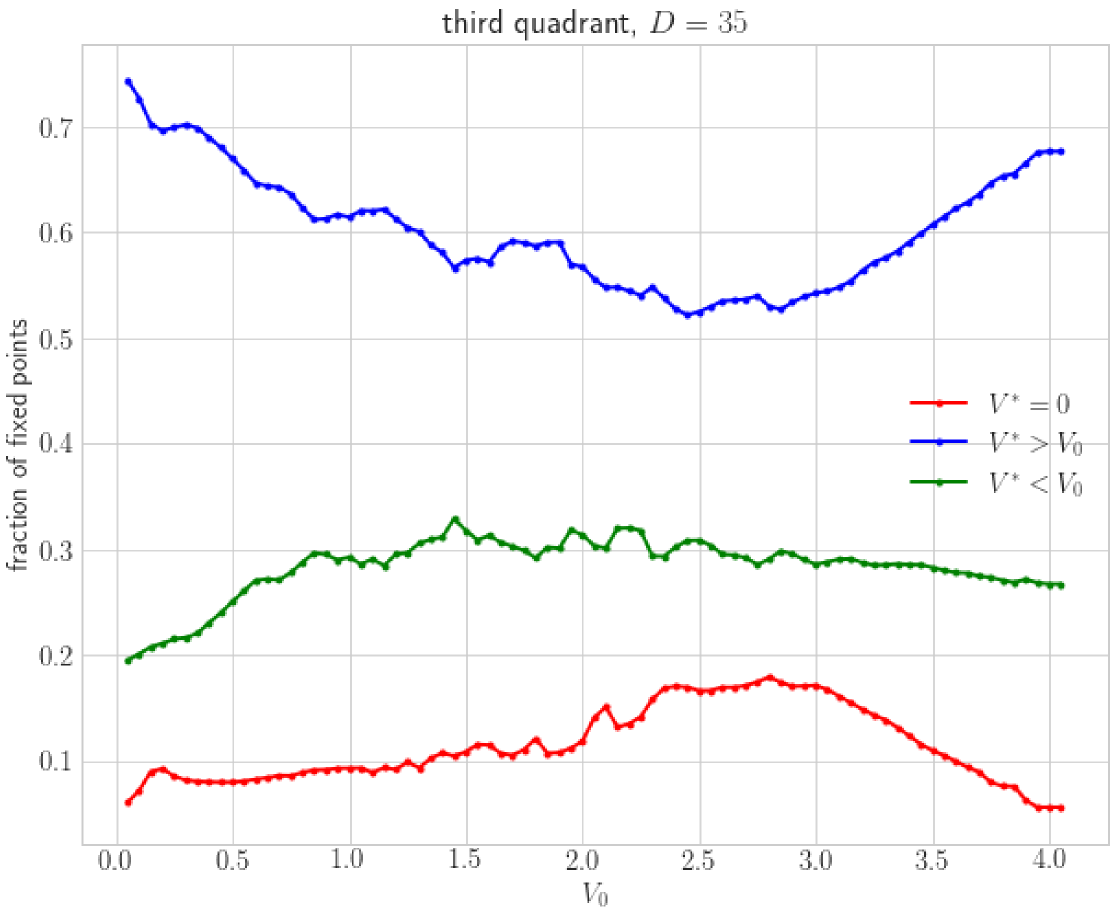
```
3.4
3.45
3.5
3.55
3.6
3.65
3.7
3.75
3.8
3.85
3.9
3.95
4.0
4.05
```



```
0.05
0.1
0.15000000000000002
0.2
```

```
0.25
0.3
0.35000000000000003
0.4
0.45
0.5
0.55
0.6000000000000001
0.6500000000000001
0.7000000000000001
0.7500000000000001
0.8
0.8500000000000001
0.9000000000000001
0.9500000000000001
1.0
1.05
1.1
1.1500000000000001
1.2000000000000002
1.2500000000000002
1.3
1.35
1.4000000000000001
1.4500000000000002
1.5000000000000002
1.55
1.6
1.6500000000000001
1.7000000000000002
1.7500000000000002
1.8
1.85
1.9000000000000001
1.9500000000000002
2.0
2.05
2.1
2.15
2.1999999999999997
2.25
2.3
2.35
2.4
2.45
2.5
2.55
2.6
```

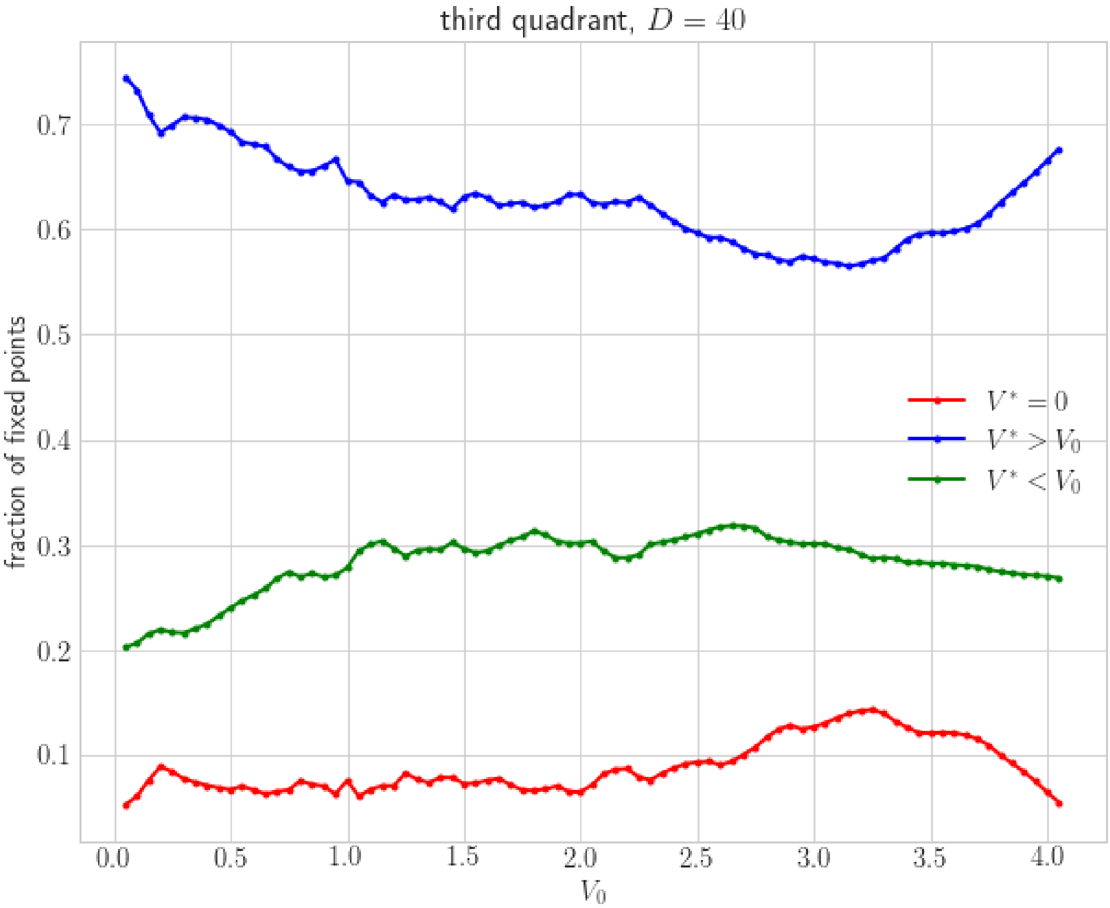
```
2.65
2.7
2.75
2.8
2.85
2.9
2.95
3.0
3.05
3.1
3.15
3.2
3.25
3.3
3.35
3.4
3.45
3.5
3.55
3.6
3.65
3.7
3.75
3.8
3.85
3.9
3.95
4.0
4.05
```



```
0.05
0.1
0.15000000000000002
0.2
0.25
0.3
0.35000000000000003
0.4
0.45
0.5
0.55
0.6000000000000001
0.6500000000000001
0.7000000000000001
0.7500000000000001
0.8
0.8500000000000001
0.9000000000000001
0.9500000000000001
```

```
1.0
1.05
1.1
1.1500000000000001
1.2000000000000002
1.2500000000000002
1.3
1.35
1.4000000000000001
1.4500000000000002
1.5000000000000002
1.55
1.6
1.6500000000000001
1.7000000000000002
1.7500000000000002
1.8
1.85
1.9000000000000001
1.9500000000000002
2.0
2.05
2.1
2.15
2.1999999999999997
2.25
2.3
2.35
2.4
2.45
2.5
2.55
2.6
2.65
2.7
2.75
2.8
2.85
2.9
2.95
3.0
3.05
3.1
3.15
3.2
3.25
3.3
3.35
```

```
3.4
3.45
3.5
3.55
3.6
3.65
3.7
3.75
3.8
3.85
3.9
3.95
4.0
4.05
```



```
[ ]: plot_all(0.2, 0.1, -1, r"fourth quadrant", V0_range=np.arange(0.01,1,0.01))
```

3.9 b. Change in the fraction of irrelevant fixed points under increase in D

Next we will see how the ratio of number of fixed points in each class varies as we increase the bandwidth, for a particular $V \sim 10$ in the stable region.

[ ]:

```
def plot_frac(J0, K0, sign, title):
    D0_range = np.arange(10,91,20)
    frac = [].05
    for D0 in D0_range:
        print (D0)
        V0 = 10
        count = (count_fp(D0, V0, J0, K0, sign))
        frac.append(count[0]/sum(count))
    plt.plot(D0_range, frac, marker=".")
    plt.title(title)
    plt.xlabel(r"$D_0$")
    plt.ylabel(r"log of ratio of irr. to rel.")
    plt.show()


#plot_frac(0.04, 0.03, 1, r"first quadrant")
#plot_frac(0.03, 0.04, 1, r"second quadrant")
#plot_frac(0.03, 0.04, -1, r"third quadrant")
#plot_frac(0.04, 0.03, -1, r"fourth quadrant")
```

3.10 c. Change in the critical V under increase in D

For the first and third quadrants, there is a critical value of V at which the number of relevant and irrelevant fixed points become equal. We will now see how this value depends on the bandwidth D.

[ ]:

```
def Vc_vs_D(J0, K0, sign, title):
    D0_range = range(10,91,20)
    Vc = [get_Vc(D0, J0, K0, sign, title) for D0 in D0_range]
    plt.plot(D0_range, Vc, marker=".")
    plt.title(title)
    plt.xlabel(r"$D_0$")
    plt.ylabel(r"$V_c$")
    plt.show()


#Vc_vs_D(0.4, 0.3, 1, r"first quadrant")
#Vc_vs_D(0.3, 0.4, -1, r"third quadrant")
```