MAPK Networks: Compute Numerical Simulation

In [1]:

```
import scipy as sp
import scipy.integrate as integrate
import numpy as np
import matplotlib.pyplot as plt
import os
import subprocess
%config InlineBackend.figure_format = 'svg'
```

In [4]:

```
def systemOfEquations(variables, time):

    x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11 = variables

    k1 = 0.02
    k2 = 1
    k3 = 0.01
    k4 = 0.032
    k5 = 1
    k6 = 15
    k7 = 0.045
    k8 = 1
    k9 = 0.092
    k10 = 1
    k11 = 0.01
    k12 = 0.01
    k13 = 1
    k14 = 0.5
    k15 = 0.086
    k16 = 0.0011

    a1 = (k2*x6) + (k15*x11) - (k1*x1*x4) - (k16*x1*x5)
    a2 = (k3*x6) + (k5*x7) + (k10*x9) + (k13*x10) - (x2*x5*(k11 + k12)) - (k4*x2*x4
    a3 = (k6*x7) + (k8*x8) - (k7*x3*x5)
    a4 = (x6*(k2 + k3)) + (x7*(k5 + k6)) - (k1*x1*x4) - (k4*x2*x4)
    a5 = (k8*x8) + (k10*x9) + (k13*x10) + (k15*x11) - (x2*x5*(k11 + k12)) - (k7*x3*
    a6 = (k1*x1*x4) - (x6*(k2 + k3))
    a7 = (k4*x2*x4) - (x7*(k5 + k6))
    a8 = (k7*x3*x5) - (x8*(k8 + k9))
    a9 = (k9*x8) - (k10*x9) + (k11*x2*x5)
    a10 = (k12*x2*x5) - (x10*(k13 + k14))
    a11 = (k14*x10) - (k15*x11) + (k16*x1*x5)

    return [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11]
```

In [5]:

```
def criteriaCheck1(variables):
    x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11 = variables
    if (x5 + x8 + x9 + x10 + x11 == 100) and (x4 + x6 + x7 == 50) and (x1 + x2 + x3
        return True
    else:
        return False
```

In [6]:

```
def criteriaCheck2(variables):
    x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11 = variables
    if (x5 + x8 + x9 + x10 + x11 == 100) and (x4 + x6 + x7 == 50) and (x1 + x2 + x3
        return True
    else:
        return False
```

In [7]:

```
def createPlotsTask7N8(solution, t):
    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), dpi=360)
    axes.plot(t, solution[:, 0], "black",
              t, solution[:, 1], "green",
              t, solution[:, 2], "red",
              t, solution[:, 3], "cyan",
              t, solution[:, 4], "magenta",
              t, solution[:, 5], "yellow",
              t, solution[:, 6], "blue",
              t, solution[:, 7], "silver",
              t, solution[:, 8], "gold",
              t, solution[:, 9], "indigo",
              t, solution[:, 10], "brown")
    plt.title("MAPK Cascade")
    plt.xlabel('Time')
    plt.ylabel('Concentration')
    plt.show()
```

In [8]:

```
def evaluateTrajectories(solution, t):
    eval1 = solution[:, 0] + solution[:, 1] + solution[:, 2] + solution[:, 3] + sol
    eval2 = solution[:, 4] + solution[:, 7] + solution[:, 8] + solution[:, 9] + sol
    eval3 = solution[:, 3] + solution[:, 5] + solution[:, 6]
    eval4 = solution[:, 0] + solution[:, 1] + solution[:, 2] + solution[:, 5] + sol
    eval5 = solution[:, 0] + solution[:, 1] + solution[:, 2] + solution[:, 3] + sol
    figeval, axeseval = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), dpi=360)
    axeseval.plot(t, eval1, "b",t, eval2, "g",t, eval3, "r",t, eval4, "c",t, eval5,
    plt.title("MAPK Cascade")
    plt.xlabel('Time')
    plt.ylabel('Concentration')
    plt.show()
```

In [9]:

```
def plotGraphsForTask7(initialCondition):
    if criteriaCheck1(initialCondition) == True:
        t_start = 0
        t_end = 7000
        t_step = 100000
        t = np.linspace(t_start, t_end, t_step)
        solution = integrate.odeint(systemOfEquations, initialCondition, t)
        createPlotsTask7N8(solution, t)
        evaluateTrajectories(solution,t)
    else:
        print("Wrong set of inputs")
```

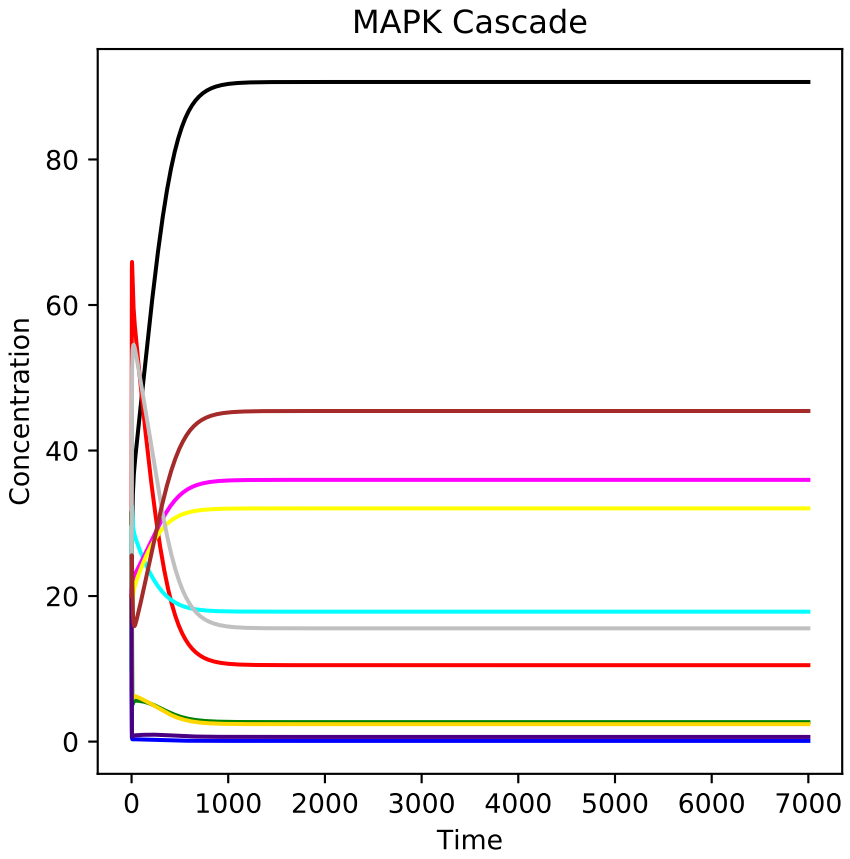
In [10]:

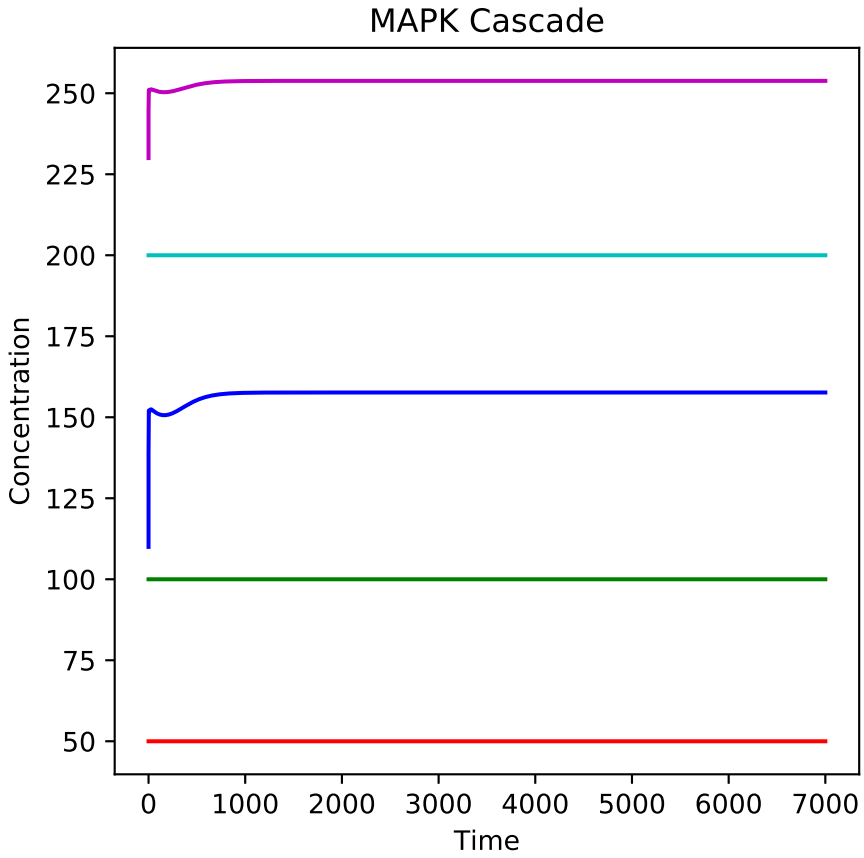
```
def plotGraphsForTask8(initialCondition):
    if criteriaCheck2(initialCondition) == True:
        t_start = 0
        t_end = 7000
        t_step = 100000
        t = np.linspace(t_start, t_end, t_step)
        solution = integrate.odeint(systemOfEquations, initialCondition, t)
        createPlotsTask7N8(solution, t)
        evaluateTrajectories(solution,t)
    else:
        print("Wrong set of inputs")
```

Task 7

In [11]:

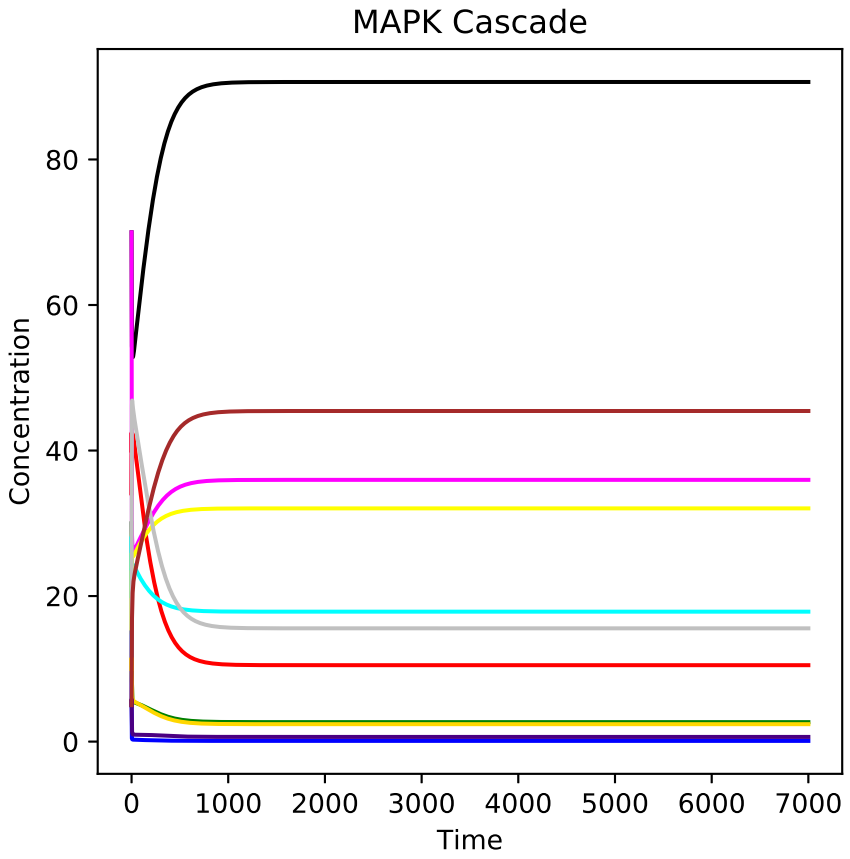
```
#Initial condition
initialCondition = [20, 30, 30, 10, 20, 20, 20, 20, 20, 20, 20]
plotGraphsForTask7(initialCondition)
```

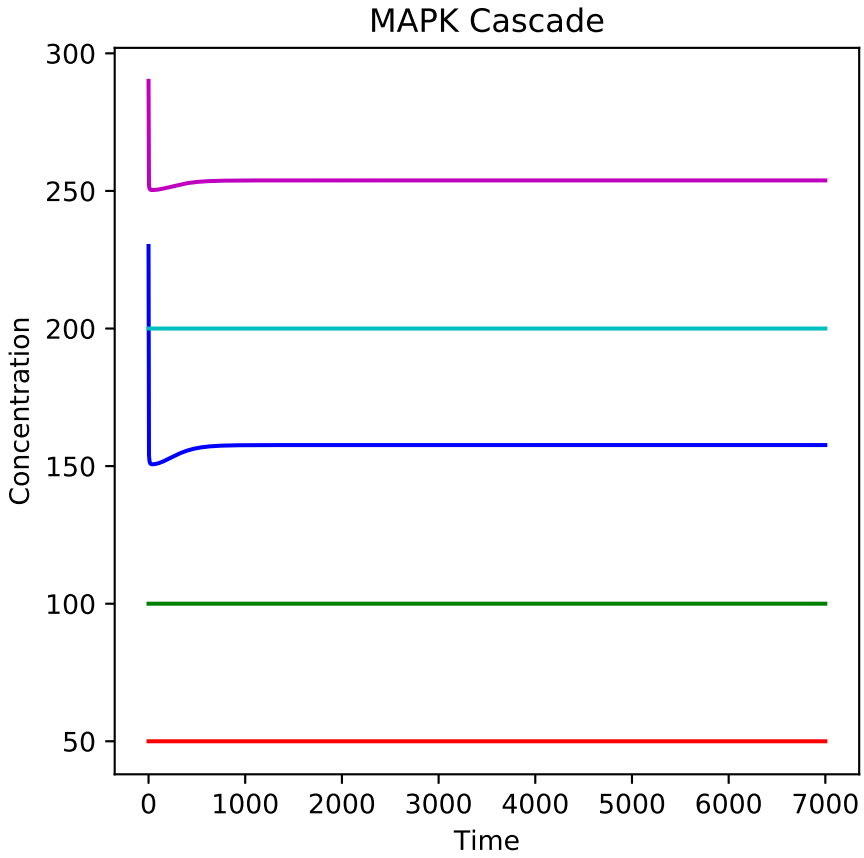




In [12]:

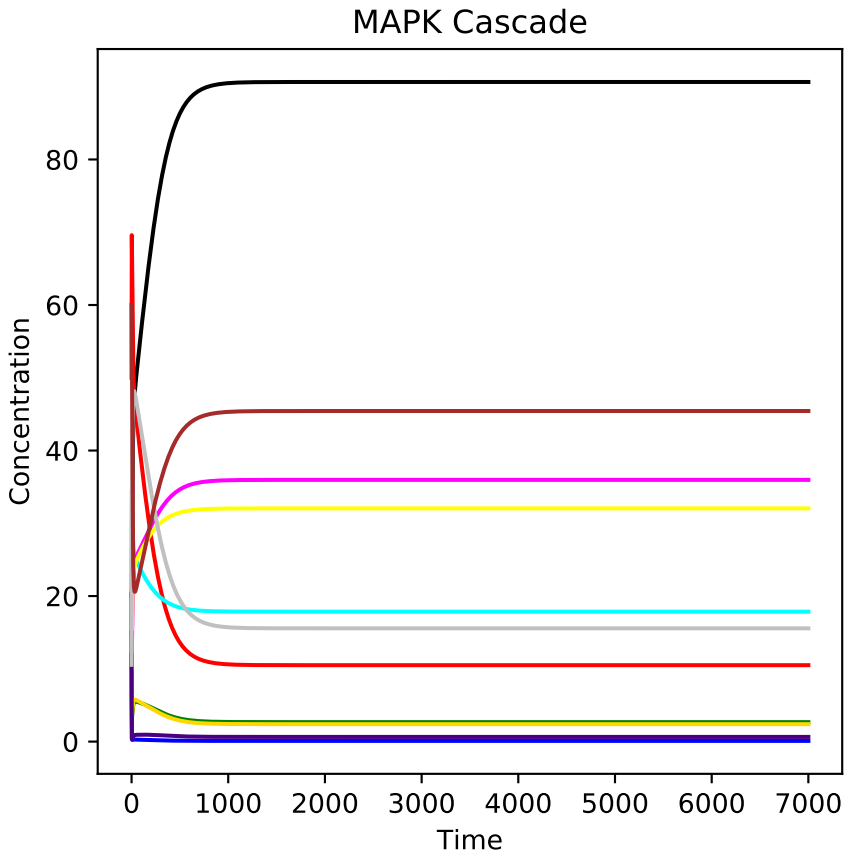
```
#Initial condition
initialCondition = [70, 30, 40, 20, 70, 15, 15, 10, 10, 5, 5]
plotGraphsForTask7(initialCondition)
```

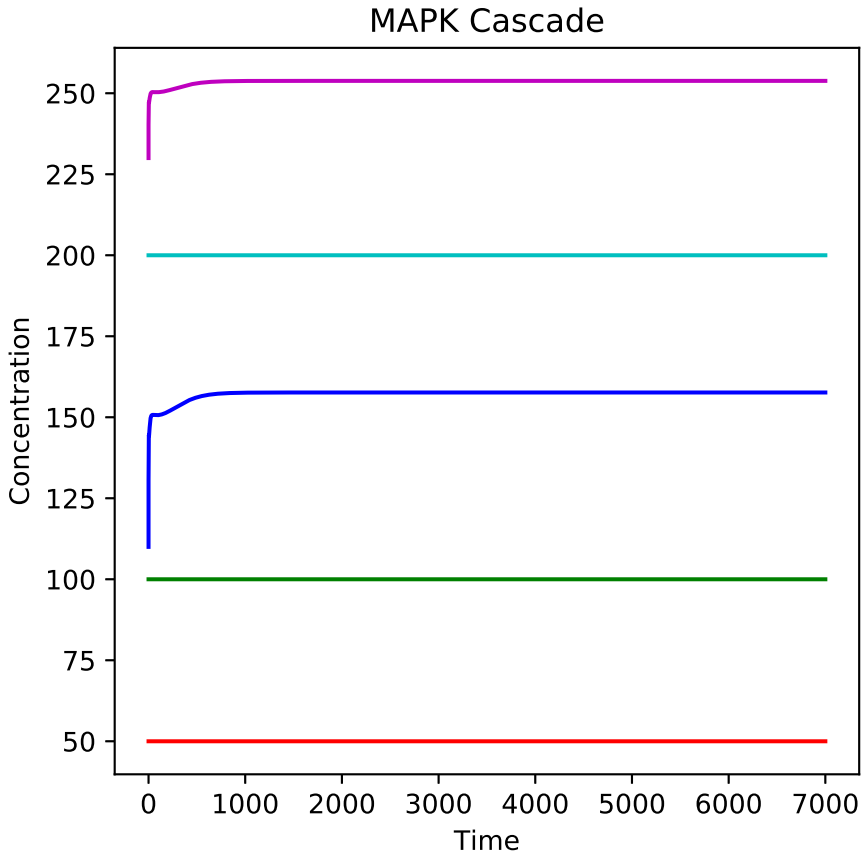




In [13]:

```
#Initial condition
initialCondition = [10, 20, 50, 20, 10, 15, 15, 10, 10, 10, 60]
plotGraphsForTask7(initialCondition)
```

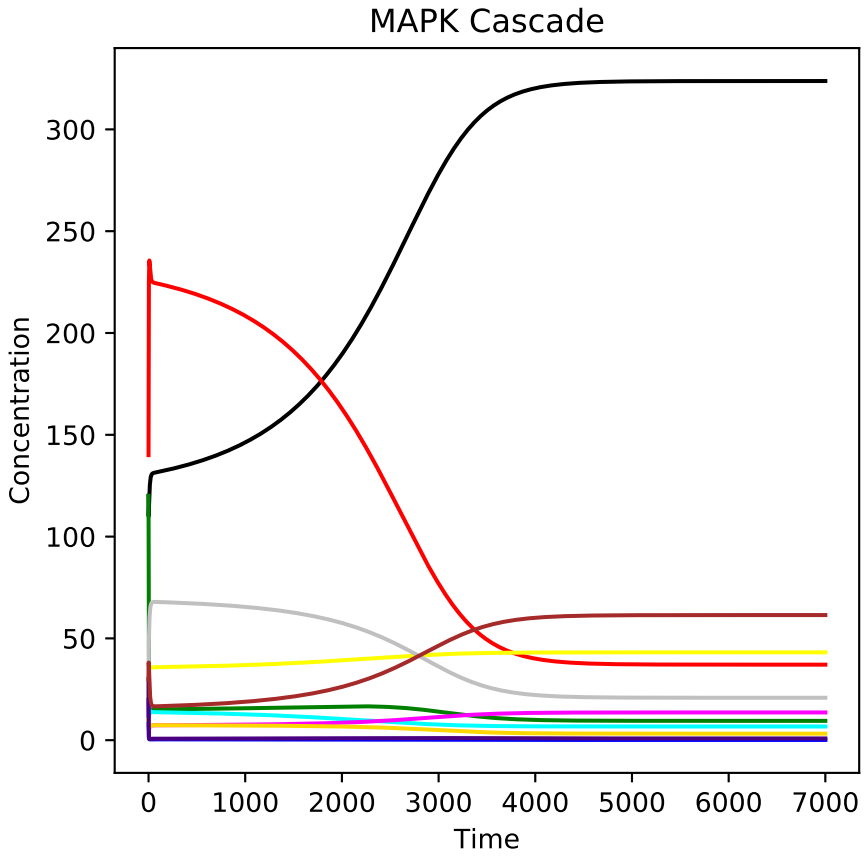


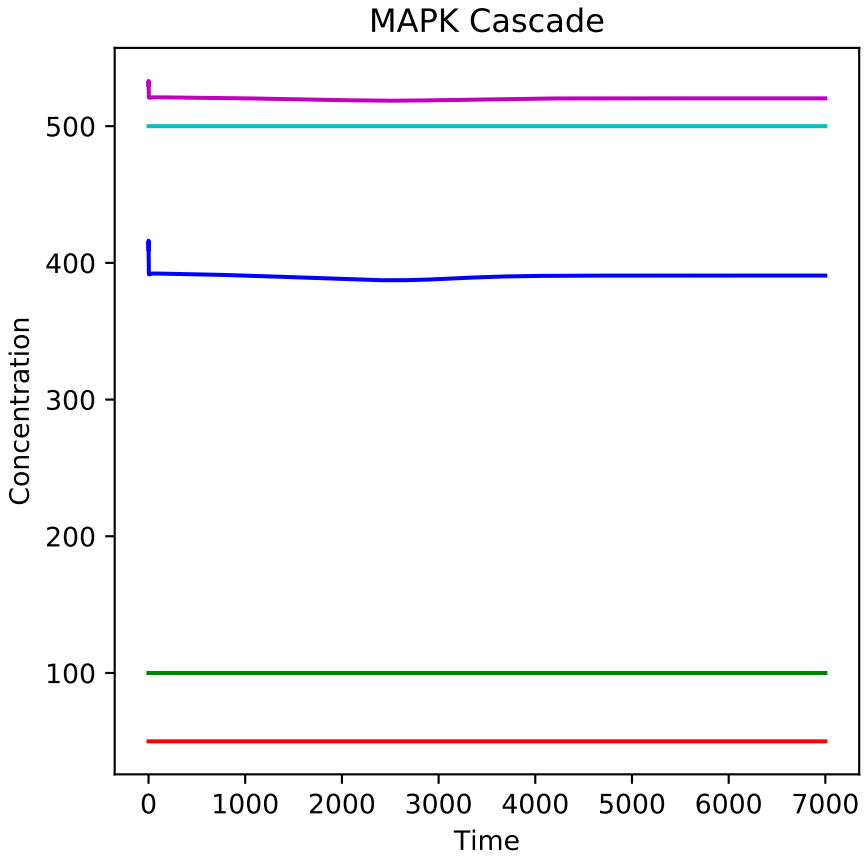


Task 8

In [14]:

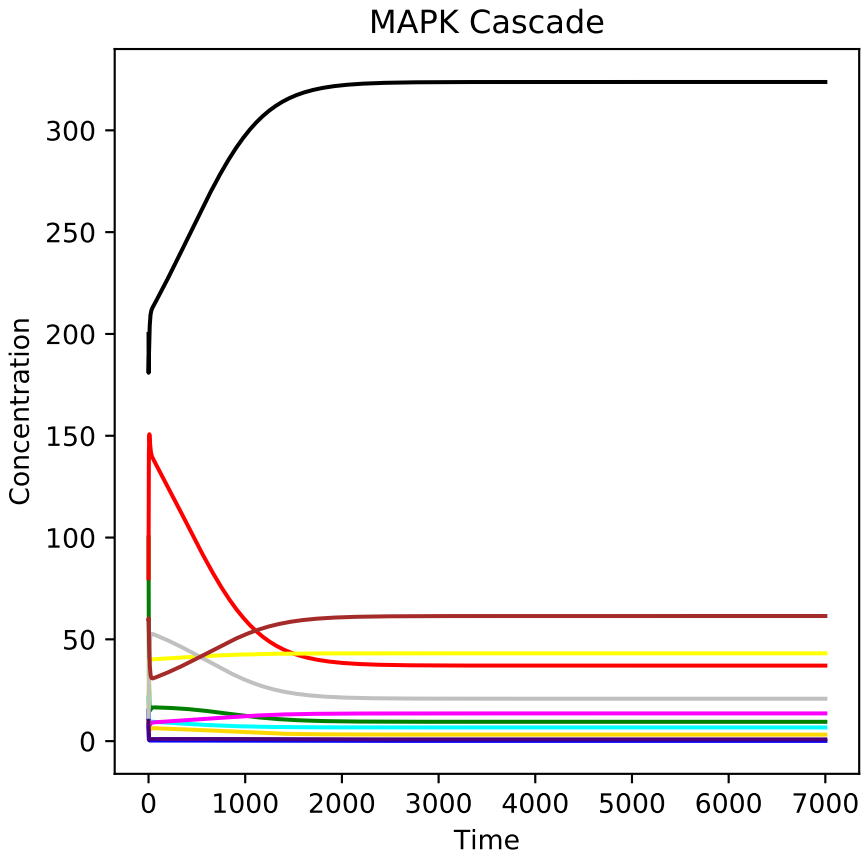
```
#Initial condition
initialCondition = [120, 120, 140, 10, 20, 20, 20, 10, 10, 30, 30]
plotGraphsForTask8(initialCondition)
```

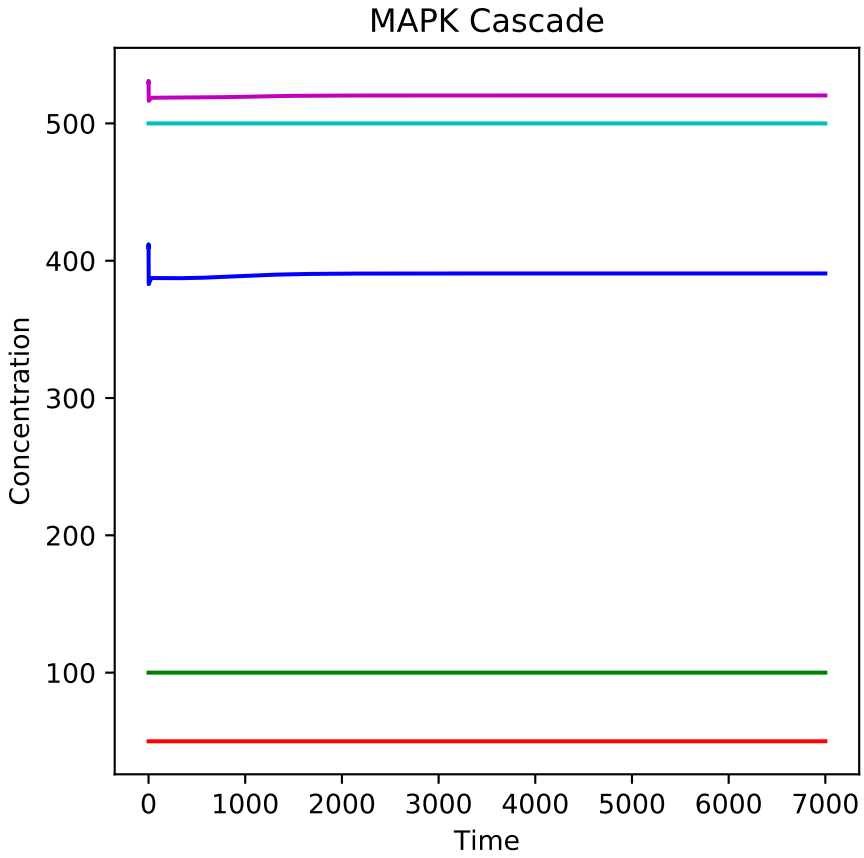




In [15]:

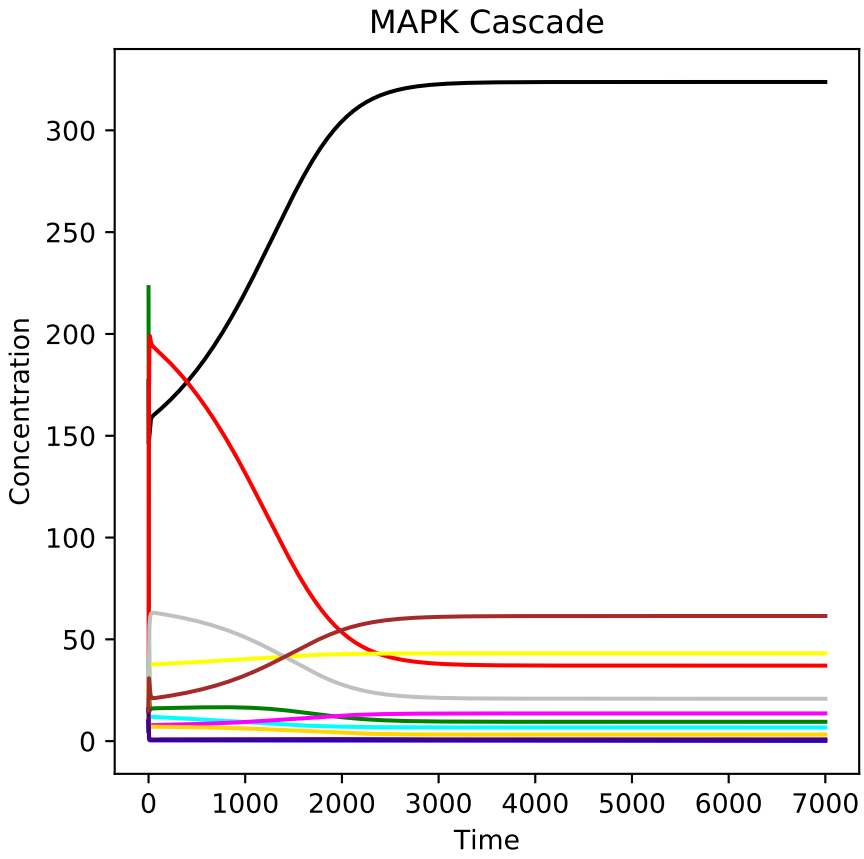
```
#Initial condition
initialCondition = [200, 100, 80, 20, 10, 15, 15, 10, 10, 10, 60]
plotGraphsForTask8(initialCondition)
```

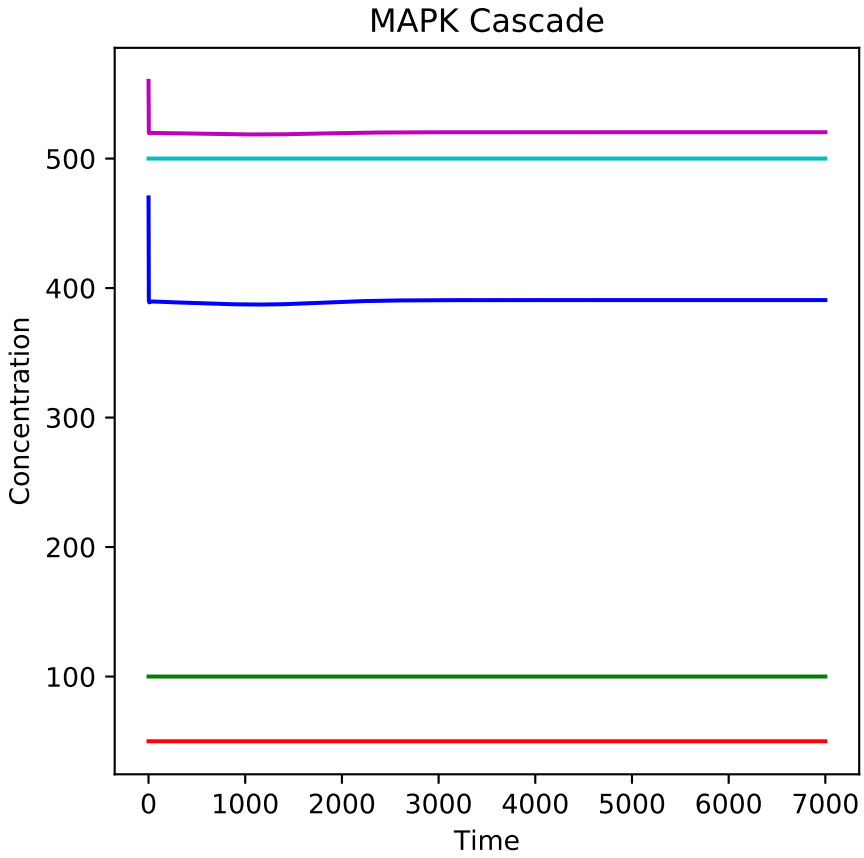




In [16]:

```
#Initial condition
initialCondition = [177, 223, 10, 40, 20, 5, 5, 25, 25, 15, 15]
plotGraphsForTask8(initialCondition)
```





Bertini

In [2]:

```
def parserBertiniOutput(file):
    lines = (line.strip() for line in file)
    nvariables = next(lines)
    dimensionSolution = int(nvariables[21:])
    variables = next(lines)
    rank = next(lines)
    skip1 = next(lines)
    dimensionTitle = next(lines)
    skip2 = next(lines)
    nonSingularTitle = next(lines)
    curline = next(lines)
    results = []
    while curline == '---------------':
        pathNumber = next(lines)
        componentNumber = next(lines)
        estimatedConditionNumber = next(lines)
        components = []
        for i in range(dimensionSolution):
            component = next(lines)
            real, imaginary = component.split(' ')
            component = float(real) + 1j * float(imaginary)
            components.append(component)
        results.append(components)
        multiplicity = next(lines)
        deflations = next(lines)
        curline = next(lines)
    return results
```

In [3]:

```
def modifiedResults(results, atol=1e-08):
    reducedResults = []
    for result in results:
        arrayForm = np.array(result)
        real = np.real(arrayForm)
        imaginary = np.imag(arrayForm)
        if np.allclose(np.zeros(shape=imaginary.shape), imaginary, atol=atol) and n
            reducedResults.append(real)
    return np.array(reducedResults)
```

In [4]:

```
#INPUT file for Bertini
bertiniInputFile = """
CONFIG
TRACKTYPE: 1;
INPUT
function y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14;
variable_group x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11;
constant k1, k2, k3, k4, k5, k6, k7, k8, k9, k10, k11, k12, k13, k14, k15, k16, k17

k1 = 0.02;
k2 = 1;
k3 = 0.01;
k4 = 0.032;
k5 = 1;
k6 = 15;
k7 = 0.045;
k8 = 1;
k9 = 0.092;
k10 = 1;
k11 = 0.01;
k12 = 0.01;
k13 = 1;
k14 = 0.5;
k15 = 0.086;
k16 = 0.0011;

k17 = 100;
k18 = 50;
k19 = {};

y1 = k2 * x6 + k15 * x11 - k1 * x1 * x4 - k16 * x1 * x5;
y2 = k3 * x6 + k5 * x7 + k10 * x9 + k13 * x10 - x2 * x5 * (k11 + k12) - k4 * x2 * x
y3 = k6 * x7 + k8 * x8 - k7 * x3 * x5;
y4 = x6 * (k2 + k3) + x7 * (k5 + k6) - k1 * x1 * x4 - k4 * x2 * x4;
y5 = k8 * x8 + k10 * x9 + k13 * x10 + k15 * x11 - x2 * x5 * (k11 + k12) - k7 * x3 *
y6 = k1 * x1 * x4 - x6 * (k2 + k3);
y7 = k4 * x2 * x4 - x7 * (k5 + k6);
y8 = k7 * x3 * x5 - x8 * (k8 + k9);
y9 = k9 * x8 - k10 * x9 + k11 * x2 * x5;
y10 = k12 * x2 * x5 - x10 * (k13 + k14);
y11 = k14 * x10 - k15 * x11 + k16 * x1 * x5;

y12 = x5 - k17 + x8 + x9 + x10 + x11;
y13 = x4 - k18 + x6 + x7;
y14 = x1 - k19 + x2 + x3 + x6 + x7 + x8 + x9 + x10 + x11;

END;
"""
```

In [5]:

```
def computation(value, pathForData, subfolderPath):
    subfolder = os.path.join(pathForData, subfolderPath.format(0))
    if not os.path.exists(subfolder):
        os.mkdir(subfolder)
    os.chdir(subfolder)
    with open(os.path.join(subfolder, 'input'), 'w') as file:
        writeData = bertiniInputFile.format(value)
        file.write(writeData)
    subprocess.call(['/home/icxa/Documents/Software/BertiniLinux64_v1.5.1/./bertini
    with open(os.path.join(subfolder, 'main_data')) as file:
        data = parserBertiniOutput(file)
        solNum = len(modifiedResults(data))
    return solNum
```

In [6]:

```
pathForData = os.path.abspath('/home/icxa/Documents/LabTasks5-16March/Data1')
if not os.path.exists(pathForData):
    os.mkdir(pathForData)
subfolderPath = 'input{:04}/'
```

In [7]:

```
k19PossibleValues = [200, 500]
for value in k19PossibleValues:
    solNum = computation(value, pathForData, subfolderPath)
    print("For k19 as ", value, "number of positive real solutions are:", solNum)
```

```
For k19 as  200 number of positive real solutions are: 1
For k19 as  500 number of positive real solutions are: 3
```

Varying K19 values upto one digit of accuracy

In [29]:

```
lowerBound = 200
upperBound = 501
interval = 1
possibleValues = np.arange(lowerBound, upperBound, interval)
```

In [30]:

```
print(len(possibleValues))
```

```
301
```

In [31]:

```
print(possibleValues)
```

```
[200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 2
17
 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 2
35
 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 2
53
 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 2
71
 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 2
89
 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 3
07
 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 3
25
 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 3
43
 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 3
61
 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 3
79
 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 3
97
 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 4
15
 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 4
33
 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 4
51
 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 4
69
 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 4
87
 488 489 490 491 492 493 494 495 496 497 498 499 500]
```

In [32]:

```
pathForData = os.path.abspath('/home/icxa/Documents/LabTasks5-16March/Data2')
if not os.path.exists(pathForData):
    os.mkdir(pathForData)
subfolderPath = 'input{:04}/'
```

In [33]:

```
results = []
for value in possibleValues:
    solNum = computation(value, pathForData, subfolderPath)
    results.append((value, solNum))
```

In [34]:

```
print(results)
```

```
[(200, 1), (201, 1), (202, 1), (203, 0), (204, 1), (205, 1), (206, 1),
(207, 1), (208, 1), (209, 1), (210, 1), (211, 1), (212, 1), (213, 1),
(214, 1), (215, 1), (216, 1), (217, 1), (218, 0), (219, 1), (220, 1),
(221, 1), (222, 1), (223, 1), (224, 1), (225, 1), (226, 1), (227, 1),
(228, 1), (229, 1), (230, 1), (231, 0), (232, 1), (233, 1), (234, 1),
(235, 1), (236, 1), (237, 1), (238, 1), (239, 1), (240, 1), (241, 1),
(242, 1), (243, 1), (244, 1), (245, 1), (246, 1), (247, 1), (248, 1),
(249, 1), (250, 1), (251, 1), (252, 1), (253, 1), (254, 1), (255, 1),
(256, 1), (257, 1), (258, 1), (259, 1), (260, 1), (261, 1), (262, 1),
(263, 1), (264, 1), (265, 1), (266, 1), (267, 1), (268, 1), (269, 1),
(270, 1), (271, 0), (272, 1), (273, 1), (274, 1), (275, 1), (276, 1),
(277, 1), (278, 1), (279, 1), (280, 1), (281, 1), (282, 1), (283, 1),
(284, 1), (285, 1), (286, 1), (287, 1), (288, 1), (289, 1), (290, 1),
(291, 1), (292, 1), (293, 1), (294, 1), (295, 1), (296, 1), (297, 1),
(298, 1), (299, 1), (300, 1), (301, 1), (302, 1), (303, 1), (304, 1),
(305, 1), (306, 1), (307, 0), (308, 1), (309, 1), (310, 1), (311, 1),
(312, 1), (313, 1), (314, 1), (315, 1), (316, 1), (317, 1), (318, 1),
(319, 1), (320, 1), (321, 1), (322, 1), (323, 1), (324, 1), (325, 1),
(326, 1), (327, 1), (328, 1), (329, 1), (330, 1), (331, 1), (332, 1),
(333, 1), (334, 1), (335, 1), (336, 1), (337, 1), (338, 1), (339, 1),
(340, 1), (341, 1), (342, 1), (343, 1), (344, 1), (345, 1), (346, 1),
(347, 1), (348, 1), (349, 1), (350, 1), (351, 1), (352, 1), (353, 1),
(354, 1), (355, 1), (356, 1), (357, 1), (358, 1), (359, 1), (360, 1),
(361, 1), (362, 1), (363, 1), (364, 1), (365, 1), (366, 1), (367, 1),
(368, 1), (369, 1), (370, 1), (371, 0), (372, 1), (373, 1), (374, 1),
(375, 1), (376, 1), (377, 1), (378, 1), (379, 1), (380, 1), (381, 1),
(382, 1), (383, 1), (384, 1), (385, 1), (386, 1), (387, 1), (388, 1),
(389, 1), (390, 1), (391, 1), (392, 1), (393, 1), (394, 1), (395, 1),
(396, 1), (397, 1), (398, 1), (399, 1), (400, 1), (401, 1), (402, 1),
(403, 1), (404, 1), (405, 1), (406, 1), (407, 1), (408, 1), (409, 1),
(410, 3), (411, 3), (412, 3), (413, 3), (414, 3), (415, 3), (416, 3),
(417, 3), (418, 3), (419, 3), (420, 3), (421, 2), (422, 3), (423, 3),
(424, 3), (425, 3), (426, 1), (427, 3), (428, 3), (429, 3), (430, 3),
(431, 3), (432, 3), (433, 3), (434, 3), (435, 3), (436, 3), (437, 3),
(438, 3), (439, 2), (440, 3), (441, 3), (442, 3), (443, 3), (444, 3),
(445, 3), (446, 3), (447, 3), (448, 3), (449, 3), (450, 3), (451, 3),
(452, 3), (453, 3), (454, 3), (455, 3), (456, 3), (457, 3), (458, 3),
(459, 3), (460, 3), (461, 3), (462, 3), (463, 1), (464, 3), (465, 3),
(466, 3), (467, 3), (468, 3), (469, 3), (470, 3), (471, 3), (472, 3),
(473, 3), (474, 3), (475, 3), (476, 3), (477, 3), (478, 3), (479, 3),
(480, 2), (481, 3), (482, 3), (483, 3), (484, 3), (485, 3), (486, 3),
(487, 3), (488, 3), (489, 1), (490, 3), (491, 3), (492, 3), (493, 3),
(494, 3), (495, 3), (496, 3), (497, 3), (498, 3), (499, 3), (500, 3)]
```

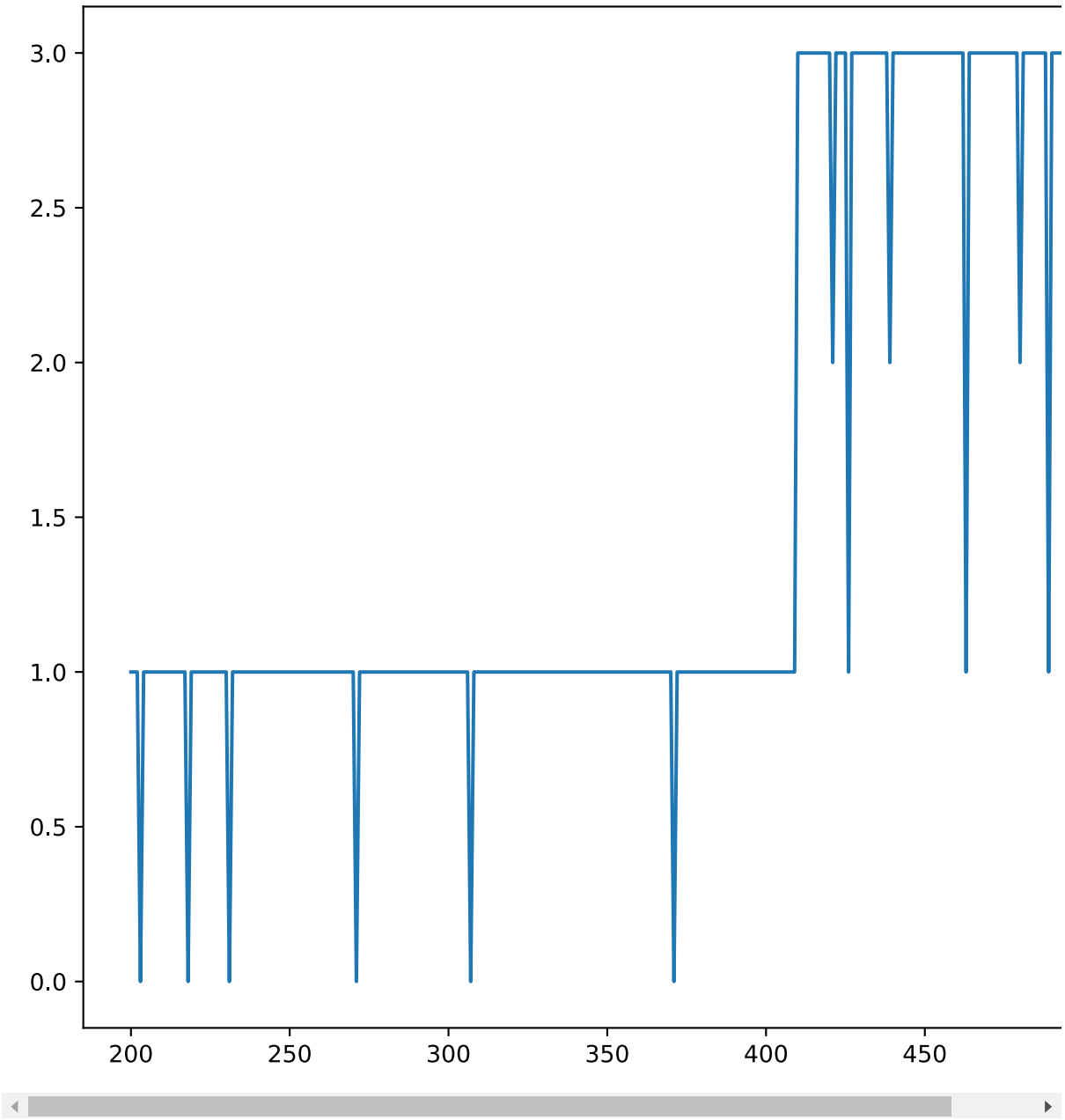
In [35]:

```
for i in range(len(results)):
    value1, solNum1 = results[i]
    for j in range(i, len(results)):
        value2, solNum2 = results[j]
        if solNum1 == 1 and solNum2 == 3 and value2-value1 == 1:
            print(value1, solNum1, value2, solNum2)
            break
```

```
409 1 410 3
426 1 427 3
463 1 464 3
489 1 490 3
```

In [37]:

```
x, y = zip(*results)
fig, axes = plt.subplots(1,1, figsize=(8, 8))
plt.plot(x, y)
plt.show()
```



Varying K19 values upto one decimal digit of accuracy

In [10]:

```
lowerBound = 200.0
upperBound = 500.1
interval = 0.1
possibleValues = np.arange(lowerBound, upperBound, interval)
```

In [11]:

```
print(len(possibleValues))
```

```
3001
```

In [12]:

```
print(possibleValues)
```

```
[ 200.   200.1  200.2 ...,  499.8  499.9  500. ]
```

In [13]:

```
pathForData = os.path.abspath('/home/icxa/Documents/LabTasks5-16March/Data4')
if not os.path.exists(pathForData):
    os.mkdir(pathForData)
subfolderPath = 'input{:04}/'
```

In [14]:

```
results = []
for value in possibleValues:
    solNum = computation(value, pathForData, subfolderPath)
    results.append((value, solNum))
```

In [15]:

```
print(results)
```

```
[(200.0, 1), (200.09999999999999, 1), (200.19999999999999, 1), (200.
29999999999998, 1), (200.39999999999998, 1), (200.49999999999997,
1), (200.59999999999997, 1), (200.69999999999996, 1), (200.799999999
99995, 1), (200.89999999999995, 1), (200.99999999999994, 1), (201.09
999999999994, 1), (201.19999999999993, 1), (201.29999999999993, 1),
(201.39999999999992, 1), (201.49999999999991, 1), (201.5999999999999
1, 0), (201.69999999999999, 1), (201.79999999999999, 1), (201.89999999
999989, 1), (201.99999999999989, 1), (202.09999999999988, 1), (202.1
9999999999987, 1), (202.29999999999987, 1), (202.39999999999986, 1),
(202.49999999999986, 1), (202.59999999999985, 1), (202.6999999999998
5, 1), (202.79999999999984, 1), (202.89999999999984, 1), (202.999999
99999983, 1), (203.09999999999982, 1), (203.19999999999982, 1), (20
3.29999999999981, 1), (203.39999999999981, 1), (203.4999999999998,
1), (203.5999999999998, 1), (203.69999999999979, 1), (203.7999999999
9978, 1), (203.89999999999978, 1), (203.99999999999977, 1), (204.099
99999999977, 1), (204.19999999999976, 1), (204.29999999999976, 1),
(204.39999999999975, 1), (204.49999999999974, 1), (204.5999999999997
4, 1), (204.69999999999973, 1), (204.79999999999973, 1), (204.899999
99999972, 1), (204.99999999999972, 1), (205.09999999999971, 1), (20
5.1999999999997, 1), (205.2999999999997, 1), (205.39999999999969
```

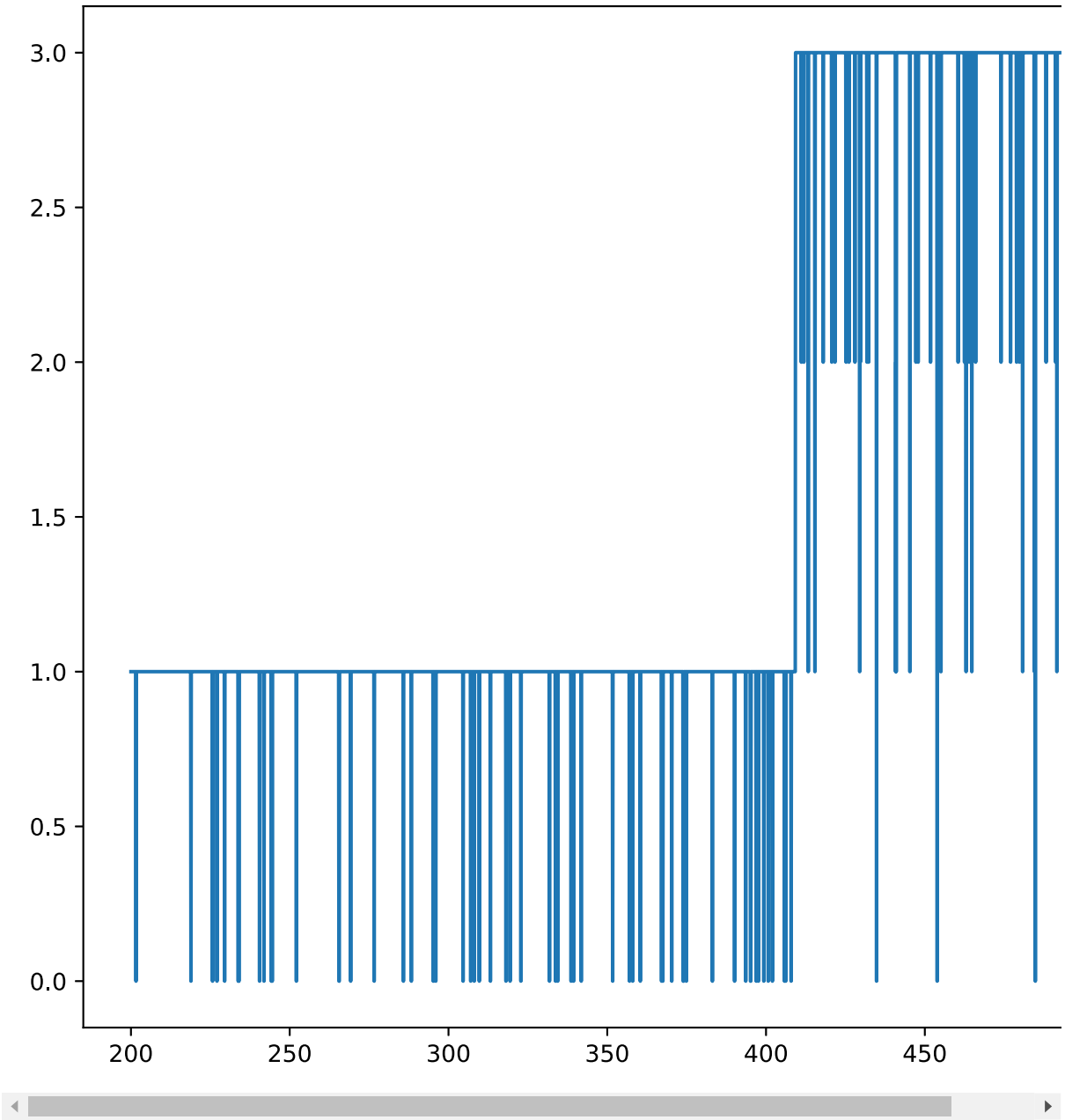
In [19]:

```
for i in range(len(results)):
    value1, solNum1 = results[i]
    for j in range(i, len(results)):
        value2, solNum2 = results[j]
        if solNum1 == 1 and solNum2 == 3 and value2-value1 < 0.2:
            print(value1, solNum1, value2, solNum2)
            break
```

```
409.2 1 409.3 3
413.3 1 413.4 3
415.4 1 415.5 3
429.5 1 429.6 3
441.0 1 441.1 3
445.3 1 445.4 3
455.0 1 455.2 3
455.1 1 455.2 3
463.0 1 463.1 3
464.8 1 464.9 3
480.8 1 480.9 3
484.5 1 484.6 3
491.6 1 491.7 3
495.1 1 495.2 3
```

In [20]:

```
x, y = zip(*results)
fig, axes = plt.subplots(1,1, figsize=(8, 8))
plt.plot(x, y)
plt.show()
```



Varying K17 in 2 orders of magnitude

In [38]:

```
lowerBound = 0
upperBound = 1001
interval = 1
possibleValuesK17 = list(np.arange(lowerBound, upperBound, interval))
```

In [39]:

```
print(possibleValuesK17)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 3
7, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 7
2, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105,
106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133,
134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147,
148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,
176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217,
218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245,
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273,
274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287,
288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301,
302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315,
316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329,
330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343,
344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357,
358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371,
372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385,
386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399,
400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413,
414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,
428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455,
456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469,
470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483,
484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497,
498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511,
512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525,
526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539,
540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553,
554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567,
568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581,
582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595,
596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609,
610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623,
624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637,
638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651,
652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665,
666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679,
680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693,
694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707,
708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721,
722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735,
736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749,
750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763,
764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777,
778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791,
792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805,
806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819,
```

```
820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833,
834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847,
848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861,
862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875,
876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889,
890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903,
904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917,
918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931,
932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945,
946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959,
960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973,
974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987,
988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000]
```

In [40]:

```
possibleValuesK19 = [200,500]
```

In [41]:

```
possibleVariations = [(k17, k19) for k17 in possibleValuesK17 for k19 in possibleVa
```

In [42]:

```
print(possibleVariations)
```

```
[(0, 200), (0, 500), (1, 200), (1, 500), (2, 200), (2, 500), (3, 20
0), (3, 500), (4, 200), (4, 500), (5, 200), (5, 500), (6, 200), (6,
500), (7, 200), (7, 500), (8, 200), (8, 500), (9, 200), (9, 500), (1
0, 200), (10, 500), (11, 200), (11, 500), (12, 200), (12, 500), (13,
200), (13, 500), (14, 200), (14, 500), (15, 200), (15, 500), (16, 20
0), (16, 500), (17, 200), (17, 500), (18, 200), (18, 500), (19, 20
0), (19, 500), (20, 200), (20, 500), (21, 200), (21, 500), (22, 20
0), (22, 500), (23, 200), (23, 500), (24, 200), (24, 500), (25, 20
0), (25, 500), (26, 200), (26, 500), (27, 200), (27, 500), (28, 20
0), (28, 500), (29, 200), (29, 500), (30, 200), (30, 500), (31, 20
0), (31, 500), (32, 200), (32, 500), (33, 200), (33, 500), (34, 20
0), (34, 500), (35, 200), (35, 500), (36, 200), (36, 500), (37, 20
0), (37, 500), (38, 200), (38, 500), (39, 200), (39, 500), (40, 20
0), (40, 500), (41, 200), (41, 500), (42, 200), (42, 500), (43, 20
0), (43, 500), (44, 200), (44, 500), (45, 200), (45, 500), (46, 20
0), (46, 500), (47, 200), (47, 500), (48, 200), (48, 500), (49, 20
0), (49, 500), (50, 200), (50, 500), (51, 200), (51, 500), (52, 20
0), (52, 500), (53, 200), (53, 500), (54, 200), (54, 500), (55, 20
0), (55, 500), (56, 200), (56, 500), (57, 200), (57, 500), (58, 20
0), (58, 500), (59, 200), (59, 500), (60, 200), (60, 500), (61, 20
```

In [43]:

```
#INPUT file for Bertini
bertiniInputFileT10 = """
CONFIG
TRACKTYPE: 1;
INPUT
function y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14;
variable_group x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11;
constant k1, k2, k3, k4, k5, k6, k7, k8, k9, k10, k11, k12, k13, k14, k15, k16, k17

k1 = 0.02;
k2 = 1;
k3 = 0.01;
k4 = 0.032;
k5 = 1;
k6 = 15;
k7 = 0.045;
k8 = 1;
k9 = 0.092;
k10 = 1;
k11 = 0.01;
k12 = 0.01;
k13 = 1;
k14 = 0.5;
k15 = 0.086;
k16 = 0.0011;

k17 = {};
k18 = 50;
k19 = {};

y1 = k2 * x6 + k15 * x11 - k1 * x1 * x4 - k16 * x1 * x5;
y2 = k3 * x6 + k5 * x7 + k10 * x9 + k13 * x10 - x2 * x5 * (k11 + k12) - k4 * x2 * x
y3 = k6 * x7 + k8 * x8 - k7 * x3 * x5;
y4 = x6 * (k2 + k3) + x7 * (k5 + k6) - k1 * x1 * x4 - k4 * x2 * x4;
y5 = k8 * x8 + k10 * x9 + k13 * x10 + k15 * x11 - x2 * x5 * (k11 + k12) - k7 * x3 *
y6 = k1 * x1 * x4 - x6 * (k2 + k3);
y7 = k4 * x2 * x4 - x7 * (k5 + k6);
y8 = k7 * x3 * x5 - x8 * (k8 + k9);
y9 = k9 * x8 - k10 * x9 + k11 * x2 * x5;
y10 = k12 * x2 * x5 - x10 * (k13 + k14);
y11 = k14 * x10 - k15 * x11 + k16 * x1 * x5;

y12 = x5 - k17 + x8 + x9 + x10 + x11;
y13 = x4 - k18 + x6 + x7;
y14 = x1 - k19 + x2 + x3 + x6 + x7 + x8 + x9 + x10 + x11;

END;
"""
```

In [44]:

```
def computationT10(value, pathForData, subfolderPath):
    subfolder = os.path.join(pathForData, subfolderPath.format(0))
    if not os.path.exists(subfolder):
        os.mkdir(subfolder)
    os.chdir(subfolder)
    with open(os.path.join(subfolder, 'input'), 'w') as file:
        writeData = bertiniInputFileT10.format(*value)
        file.write(writeData)
    subprocess.call(['/home/icxa/Documents/Software/BertiniLinux64_v1.5.1/./bertini
    with open(os.path.join(subfolder, 'main_data')) as file:
        data = parserBertiniOutput(file)
        solNum = len(modifiedResults(data))
    return solNum
```

In [45]:

```
pathForData = os.path.abspath('/home/icxa/Documents/LabTasks5-16March/Data3')
if not os.path.exists(pathForData):
    os.mkdir(pathForData)
subfolderPath = 'inputFolder/'
```

In [46]:

```
results = []
for value in possibleVariations:
    solNum = computationT10(value, pathForData, subfolderPath)
    results.append((value[0], value[1], solNum))
```

In [47]:

```
print(results)
```

```
[(0, 200, 0), (0, 500, 0), (1, 200, 1), (1, 500, 1), (2, 200, 1),
(2, 500, 1), (3, 200, 1), (3, 500, 1), (4, 200, 1), (4, 500, 1), (5,
200, 1), (5, 500, 1), (6, 200, 1), (6, 500, 1), (7, 200, 1), (7, 50
0, 1), (8, 200, 1), (8, 500, 1), (9, 200, 1), (9, 500, 1), (10, 200,
1), (10, 500, 1), (11, 200, 1), (11, 500, 1), (12, 200, 1), (12, 50
0, 1), (13, 200, 1), (13, 500, 1), (14, 200, 1), (14, 500, 1), (15,
200, 1), (15, 500, 1), (16, 200, 1), (16, 500, 1), (17, 200, 1), (1
7, 500, 1), (18, 200, 1), (18, 500, 1), (19, 200, 1), (19, 500, 1),
(20, 200, 1), (20, 500, 1), (21, 200, 1), (21, 500, 1), (22, 200,
1), (22, 500, 1), (23, 200, 1), (23, 500, 1), (24, 200, 1), (24, 50
0, 1), (25, 200, 1), (25, 500, 1), (26, 200, 1), (26, 500, 1), (27,
200, 1), (27, 500, 1), (28, 200, 1), (28, 500, 1), (29, 200, 1), (2
9, 500, 1), (30, 200, 1), (30, 500, 1), (31, 200, 1), (31, 500, 1),
(32, 200, 1), (32, 500, 1), (33, 200, 1), (33, 500, 1), (34, 200,
1), (34, 500, 1), (35, 200, 1), (35, 500, 1), (36, 200, 1), (36, 50
0, 1), (37, 200, 1), (37, 500, 0), (38, 200, 1), (38, 500, 1), (39,
200, 1), (39, 500, 1), (40, 200, 1), (40, 500, 1), (41, 200, 1), (4
1, 500, 1), (42, 200, 1), (42, 500, 1), (43, 200, 1), (43, 500, 1),
(44, 200, 1), (44, 500, 1), (45, 200, 1), (45, 500, 1), (46, 200,
1), (46, 500, 1), (47, 200, 1), (47, 500, 1), (48, 200, 1), (48, 50
```

In [49]:

```
x, y, z = zip(*results)
print(x,y,z)
```

```
(0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10,
11, 11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19,
19, 20, 20, 21, 21, 22, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 27,
28, 28, 29, 29, 30, 30, 31, 31, 32, 32, 33, 33, 34, 34, 35, 35, 36,
36, 37, 37, 38, 38, 39, 39, 40, 40, 41, 41, 42, 42, 43, 43, 44, 44,
45, 45, 46, 46, 47, 47, 48, 48, 49, 49, 50, 50, 51, 51, 52, 52, 53,
53, 54, 54, 55, 55, 56, 56, 57, 57, 58, 58, 59, 59, 60, 60, 61, 61,
62, 62, 63, 63, 64, 64, 65, 65, 66, 66, 67, 67, 68, 68, 69, 69, 70,
70, 71, 71, 72, 72, 73, 73, 74, 74, 75, 75, 76, 76, 77, 77, 78, 78,
79, 79, 80, 80, 81, 81, 82, 82, 83, 83, 84, 84, 85, 85, 86, 86, 87,
87, 88, 88, 89, 89, 90, 90, 91, 91, 92, 92, 93, 93, 94, 94, 95, 95,
96, 96, 97, 97, 98, 98, 99, 99, 100, 100, 101, 101, 102, 102, 103, 1
03, 104, 104, 105, 105, 106, 106, 107, 107, 108, 108, 109, 109, 110,
110, 111, 111, 112, 112, 113, 113, 114, 114, 115, 115, 116, 116, 11
7, 117, 118, 118, 119, 119, 120, 120, 121, 121, 122, 122, 123, 123,
124, 124, 125, 125, 126, 126, 127, 127, 128, 128, 129, 129, 130, 13
0, 131, 131, 132, 132, 133, 133, 134, 134, 135, 135, 136, 136, 137,
137, 138, 138, 139, 139, 140, 140, 141, 141, 142, 142, 143, 143, 14
4, 144, 145, 145, 146, 146, 147, 147, 148, 148, 149, 149, 150, 150,
```

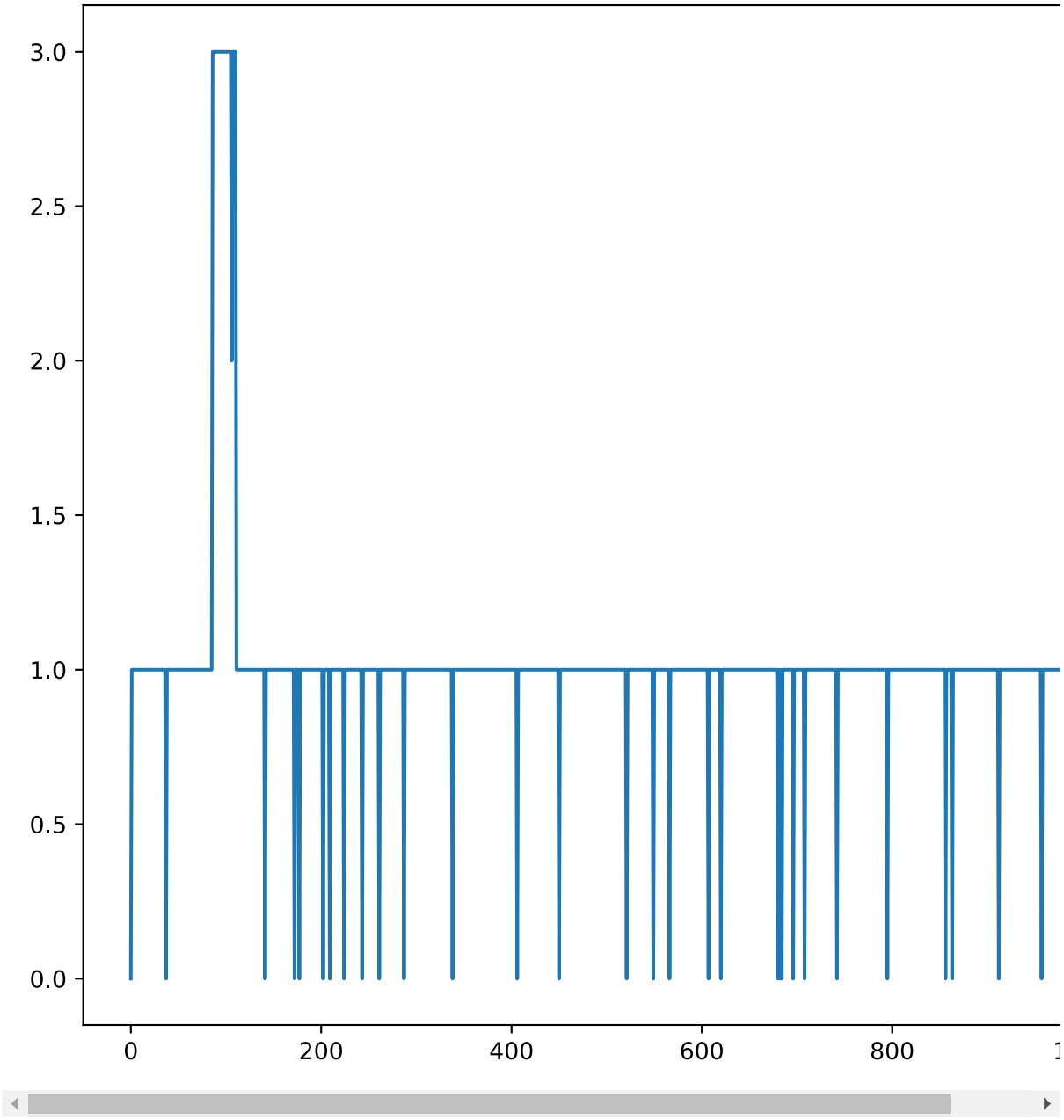
In [52]:

```
storeV1k17 = []
storeV1numSol = []
storeV2k17 = []
storeV2numSol = []
for i in range(len(results)):
    k17 = results[i][0]
    k19 = results[i][1]
    numSol = results[i][2]
    if k19 == 500:
        storeV1k17.append(k17)
        storeV1numSol.append(numSol)
    elif k19 == 200:
        storeV2k17.append(k17)
        storeV2numSol.append(numSol)
print(len(storeV1k17))
print(len(storeV2k17))
```

```
1001
1001
```

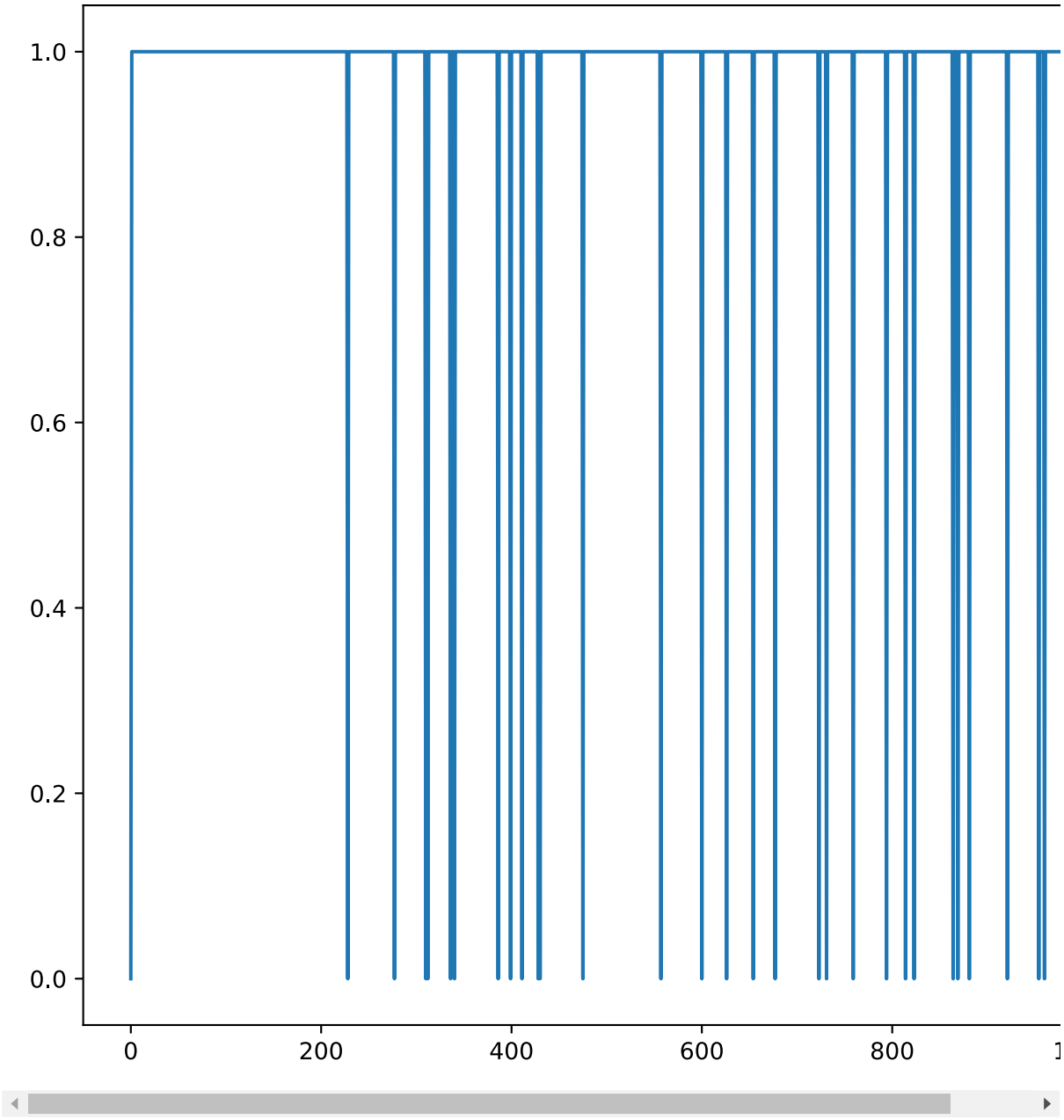
In [55]:

```
fig, axes = plt.subplots(1,1, figsize=(8, 8))
plt.plot(storeV1k17, storeV1numSol)
plt.show()
```



In [56]:

```
fig, axes = plt.subplots(1,1, figsize=(8, 8))
plt.plot(storeV2k17, storeV2numSol)
plt.show()
```



In [ ]: