Tuples and lists

- tuples
- lists
- mutability
- list comprehension
- for-if, for-for
- list()
- any(), all()
- enumerate(), zip()

Tuples

(value$_1$, value$_2$, … , value$_k$)

- Tuples can contain a sequence of zero or more elements, enclosed by "(" and ")"
- Tuples are immutable
- Tuple of length 0: ()
- Tuple of length 1: (value,)

  Note the comma to make a tuple of length one distinctive from an expression in parenthesis
- In many contexts a tuple with ≥ 1 elements can be written without parenthesis
- Accessors to lists also apply to tuples, slices, ...

Python shell

```
> (1, 2, 3)
| (1, 2, 3)
> ()
| ()
> (42)
| 42
> (42,)
| (42,)
> 1, 2
| (1, 2)
> 42,
| (42,)
> x = (3, 7)
> x
| (3, 7)
> x = 4, 6
> x
| (4, 6)
> x[1] = 42
| TypeError: 'tuple' object does
  not support item assignment
```

Question – What value is `((42,))` ?

a) `42`

b) `(42)`

c) `(42,)`

d) `((42,),)`

e) Don't know

Question – What is `x` ?

```
x = [1, [2, 3], (4, 5)]
x[2][0] = 42
```

a) `[1, [42, 3], (4, 5)]`

b) `[1, [2, 3], (42, 5)]`

c) `[1, [2, 3], 42]`

d) TypeError

e) Don’t know

Question – What tree is `('A',(('B','C'),'D'))` ?

```
          /\
         /  \
        /    \
       /      \
      /\      'D'
     /  \
    /    \
   /\    'C'
  /  \
'A'  'B'
```

a)

```
            /\
           /  \
          /    \
         /      \
        /        \
       /\        'D'
      /  \
     /    \
   'A'    /\
         /  \
       'B'  'C'
```

b)

```
          /\
         /  \
        /    \
       /      \
      /        \
     /\        /\
    /  \      /  \
  'A'  'B'  'C'  'D'
```

c)

```
          /\
         /  \
        /    \
       /      \
      /        \
    'A'        /\
              /  \
             /    \
            /\    'D'
           /  \
         'B'  'C'
```

d)

```
         /\
        /  \
       /    \
      /      \
    'A'      /\
            /  \
           /    \
         'B'    /\
               /  \
             'C'  'D'
```

e)

f) Don’t know

Tuple assignment

```
Python shell
> point = (10, 25)
> x, y = point
> x
| 10
> y
| 25
```

- Parallel assignments

  ```
  x, y, z = a, b, c
  ```

  is a short hand for a tuple assignment (right side is a single tuple)

  ```
  (x, y, z) = (a, b, c)
  ```

- First the right-hand side is evaluated completely, and then the individual values of the tuple are assigned to `x`, `y`, `z` left-to-right (length must be equal on both sides)

Nested tuple/lists assignments

- Let hand side can be nested (great for unpacking data)

```
(x, (y, (a[0], w)), a[1])
   = 1, (2, (3, 4)), 5
```

- [...] and (...) on left side matches both lists and tuples of equal length (but likely you would like to be consistent with type of parenthesis)

Python shell

```
> two_points = [(10, 25), (30, 40)]
> (x1, y1, x2, y2) = two_points
| ValueError: not enough values to
  unpack (expected 4, got 2)
> ((x1, y1), (x2, y2)) = two_points
> a = [None, None]
> v = ((2, (3, 4)), 5)
> ((y, (a[0], w)), a[1]) = v
> a
| [3, 5]
> [x, y, z] = (3, 5, 7)
> (x, y, z) = [3, 5, 7]
> [x, (y, z), w] = (1, [2, 3], 4)
> [x, (y, z), w] = (1, [2, (5, 6)], 4)
> z
| (5, 6)
```

Tuples vs lists: `a += b`

- Lists
  Extends existing list, i.e. same as `a.extend(b)`

- Tuples
  Must create a new tuple `a + b` and assign to `a`
  (since tuples are immutable)

Python shell

```
> x = [1, 2]
> y = x
> y += [3, 4]
> x
| [1, 2, 3, 4]
> y
| [1, 2, 3, 4]

> x = (1, 2)
> y = x
> y += (3, 4)
> x
| (1, 2)
> y
| (1, 2, 3, 4)
```

*variable assignment

- For a tuple of variable length a single **variable name* on the left side will be assigned a list of the remaining elements not matched by variables preceding/following *
- *Example*

```
a, *b, c = t
```

is equivalent to

```
a = t[0]
b = t[1:-1]
c = t[-1]
```

- There can be a single * in a left-hand-side tuple (but one new * in each nested tuple)

Python shell

```
> (a,*b,c,d) = (1,2,3,4,5,6)
> b
| [2, 3, 4]
> (a,*b,c,d) = (1,2,3)
> b
| []
> (a,*b,c,d) = (1,2)
| ValueError: not enough values to
  unpack (expected at least 3, got 2)
> v = ((1,2,3),4,5,6,(7,8,9,10))
> ((a,*b),*c,(d,*e)) = v
> b
| [2, 3]
> c
| [4, 5, 6]
> e
| [8, 9, 10]
```

Question – What is `b` ?

```
(*a,(b,),c) = ((1,2),((3,4)),((5,)),(6))
```

a) `(1,2)`

b) `(3,4)`

c) `5`

d) `(5,)`

e) `(6)`

f) Don't know

List comprehension (cool stuff)

- Example:

```
[ x*x for x in [1, 2, 3]]
```

returns

```
[1, 4, 9]
```

- General

[*expression* for *variable* in *sequence*]

returns a list, where *expression* is computed for each element in *sequence* assigned to *variable*

Python shell

```
> [2*x for x in [1,2,3]]
| [2, 4, 6]
> [2*x for x in range(10,15)]
| [20, 22, 24, 26, 28]
> [2*x for x in "abc"]
| ['aa', 'bb', 'cc']
> [(None, None) for _ in range(2)]
| [(None, None), (None, None)]
```

List comprehension (more cool stuff)

- Similarly to the left-hand-side in assignments, the variable part can be a (nested) tuple of variables for unpacking elements:

[*expression* for *tuple of variables* in *sequence*]

Python shell

```
> points = [(3, 4), (2, 5), (4, 7)]
> [(x, y, x*y) for (x, y) in points]
| [(3, 4, 12), (2, 5, 10), (4, 7, 28)]
> [(x, y, x*y) for x, y in points]
| [(3, 4, 12), (2, 5, 10), (4, 7, 28)]
> [x, y, x*y for (x, y) in points]
| SyntexError: invalid syntax
```

parenthesis required for the constructed tuples

List comprehension – for-if and multiple for

- List comprehensions can have nested for-loops

$$[\mathit{expression}\ \texttt{for}\ v_1\ \texttt{in}\ s_1\ \texttt{for}\ v_2\ \texttt{in}\ s_2\ \texttt{for}\ v_3\ \texttt{in}\ s_3]$$

- Can select a subset of the elements by adding an if-condition

$$[\mathit{expression}\ \texttt{for}\ v_1\ \texttt{in}\ s_1\ \texttt{if}\ \mathit{condition}]$$

- and be combined...

Python shell

```
> [(x, y) for x in range(1, 3) for y in range(4, 6)]
| [(1, 4), (1, 5), (2, 4), (2, 5)]
> [x for x in (1, 2) for x in (4, 5)]
| [4, 5, 4, 5]
> [x for x in range(1, 101) if x % 7 == 1 and x % 5 == 2]
| [22, 57, 92]
> [(x, y, x*y) for x in range(1, 11) if 6 <= x <= 7 for y in range(x, 11) if 6 <= y <= 7 and not x == y]
| [(6, 7, 42)]
```

Question – What will print the same?

```
points = [(3,7), (4,10),(12,3), (9,11), (7,5)]
print([(x, y) for x, y in points if x < y])
```

a) `print([x, y for x, y in points if x < y])`

b) `print([(x, y) for p in points if p[0] < p[1]])`

c) `print([p for p in points if p[0] < p[1]])`

d) `print([[x, y] for x, y in points if x < y])`

e) Don’t know

any, all

- `any(L)` checks if at least one element in the sequence `L` is true (list, sequence, strings, ranges, ...)

  `any([False, True, False])`

- `all(L)` checks if all elements in the sequence `L` are true

  `all([False, False, True])`

- `any` and `all` returns `True` or `False`

Python shell

```
> any((False, True, False))
| True
> any([False, False, False])
| False
> any([])
| False
> all([False, False, True])
| False
> all((True, True, True))
| True
> all(())
| True
> L = (7, 42, 13)
> any([x == 42 for x in L])
| True
> all([x == 42 for x in L])
| False
```

enumerate

`list(enumerate(L))`

returns

`[(0,L[0]), (1,L[1]), ..., (len(L)-1,L[-1])]`

Python shell

```
> points = [(1,2),(3,4),(5,6)]
> [(idx, x*y) for idx, (x, y) in enumerate(points)]
| [(0, 2), (1, 12), (2,30)]
> L = ('a','b','c')
> list(enumerate(L))
| [(0, 'a'), (1, 'b'), (2, 'c')]
> L_ = []
> for idx in range(len(L)):
>       L_.append((idx, L[idx]))
> print(L_)
| [(0, 'a'), (1, 'b'), (2, 'c')]
> list(enumerate(['a', 'b', 'c'], start=7))
| [(7, 'a'), (8, 'b'), (9, 'c')]
```

zip

`list(zip(`L_1`,`L_2`,...,`L_k`))` = `[(`$L_1[0], L_2[0], \ldots, L_k[0]$`),...,(`$L_1[n], L_2[n], \ldots, L_k[n]$`)]`

where $n = \min(\text{len}(L_1), \text{len}(L_2), \ldots, \text{len}(L_k))$

- Example ("matrix transpose"):

```
list(zip([1,2,3],
         [4,5,6],
         [7,8,9]))
```

returns

```
[(1, 4, 7),
 (2, 5, 8),
 (3, 6, 9)]
```

Python shell

```
> x = [1, 2, 3]
> y = [4, 5, 6]
| zip(x, y)
> <zip at 0xb02b530>
> points = list(zip(x, y))
> print(points)
| [(1, 4), (2, 5), (3, 6)]
```

Python shell

```
> first = ['Donald', 'Mickey', 'Scrooge']
> last = ['Duck', 'Mouse', 'McDuck']

> for i, (a, b) in enumerate(zip(first, last), start=1):
>     print(i, a, b)

| 1 Donald Duck
| 2 Mickey Mouse
| 3 Scrooge McDuck
```

(Simple) functions

- You can define your own functions using:

  `def` *function-name* `(`*var*$_1$, ..., *var*$_k$`):`

  body code

- If the body code executes

  `return` *expression*

  the result of *expression* will be returned by the function. If expression is omitted or the body code terminates without performing `return`, then `None` is returned.

- When *calling* a function *name* `(`*value*$_1$,..., *value*$_k$`)` body code is executed with *var*$_i$=*value*$_i$

Python shell

```
> def sum3(x, y, z):
      return x + y + z

> sum3(1, 2, 3)
| 6
> sum3(5, 7, 9)
| 21

> def powers(L, power):
      P = [x**power for x in L]
      return P

> powers([2, 3, 4], 3)
| [8, 27, 64]
```

Question – What tuple is printed ?

```
def even(x):
    if x % 2 == 0:
        return True
    else:
        return False

print((even(7), even(6)))
```

a) `(False, False)`

b) `(False, True)`

c) `(True, False)`

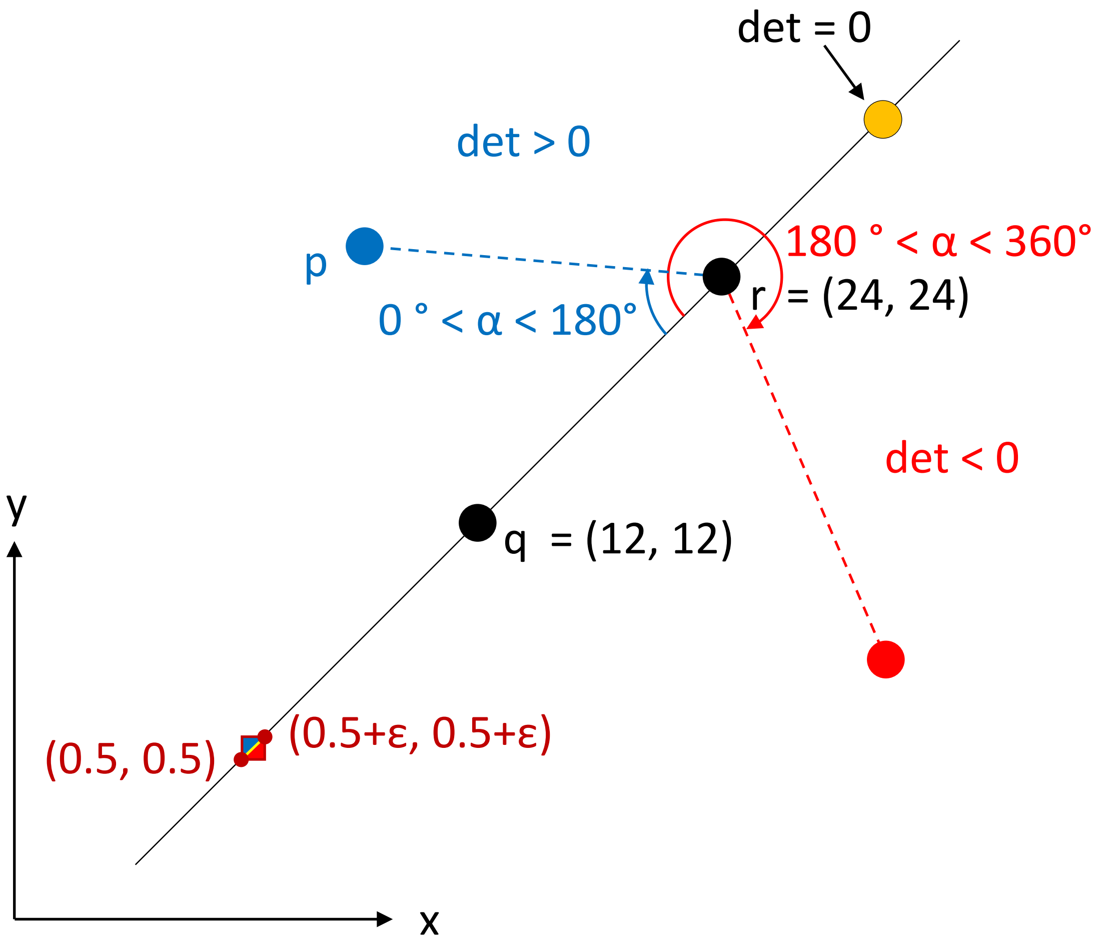
d) `(True, True)`

e) Don’t know

Geometric orientation test

Purpose of example

- illustrate tuples
- list comprehension
- matplotlib.pyplot
- floats are strange ⚠



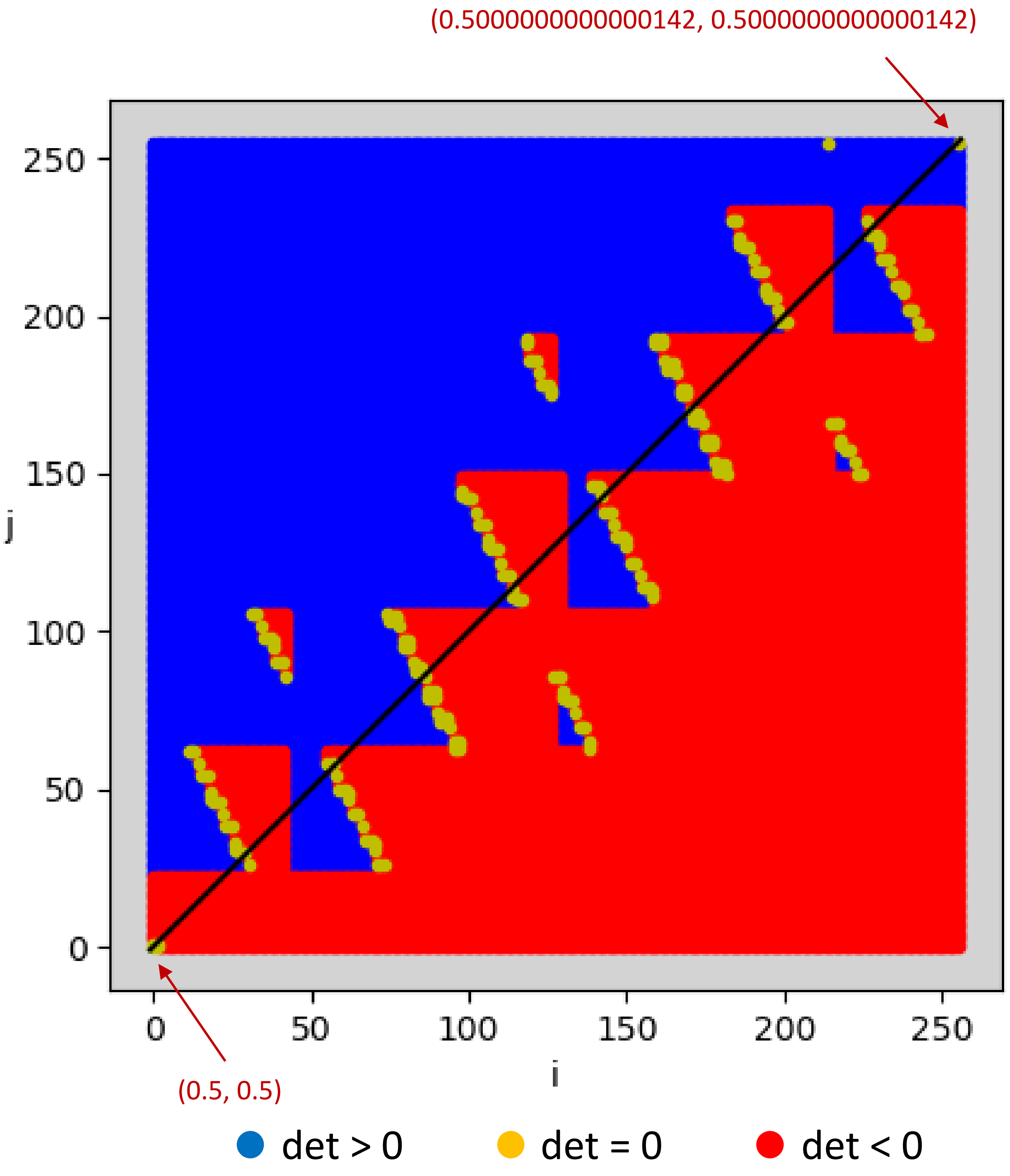
$$\det = \begin{vmatrix} 1 & q_x & q_y \\ 1 & r_x & r_y \\ 1 & p_x & p_y \end{vmatrix} = \underbrace{r_x p_y - p_x r_y - q_x p_y + p_x q_y + q_x r_y - r_x q_y}_{}$$

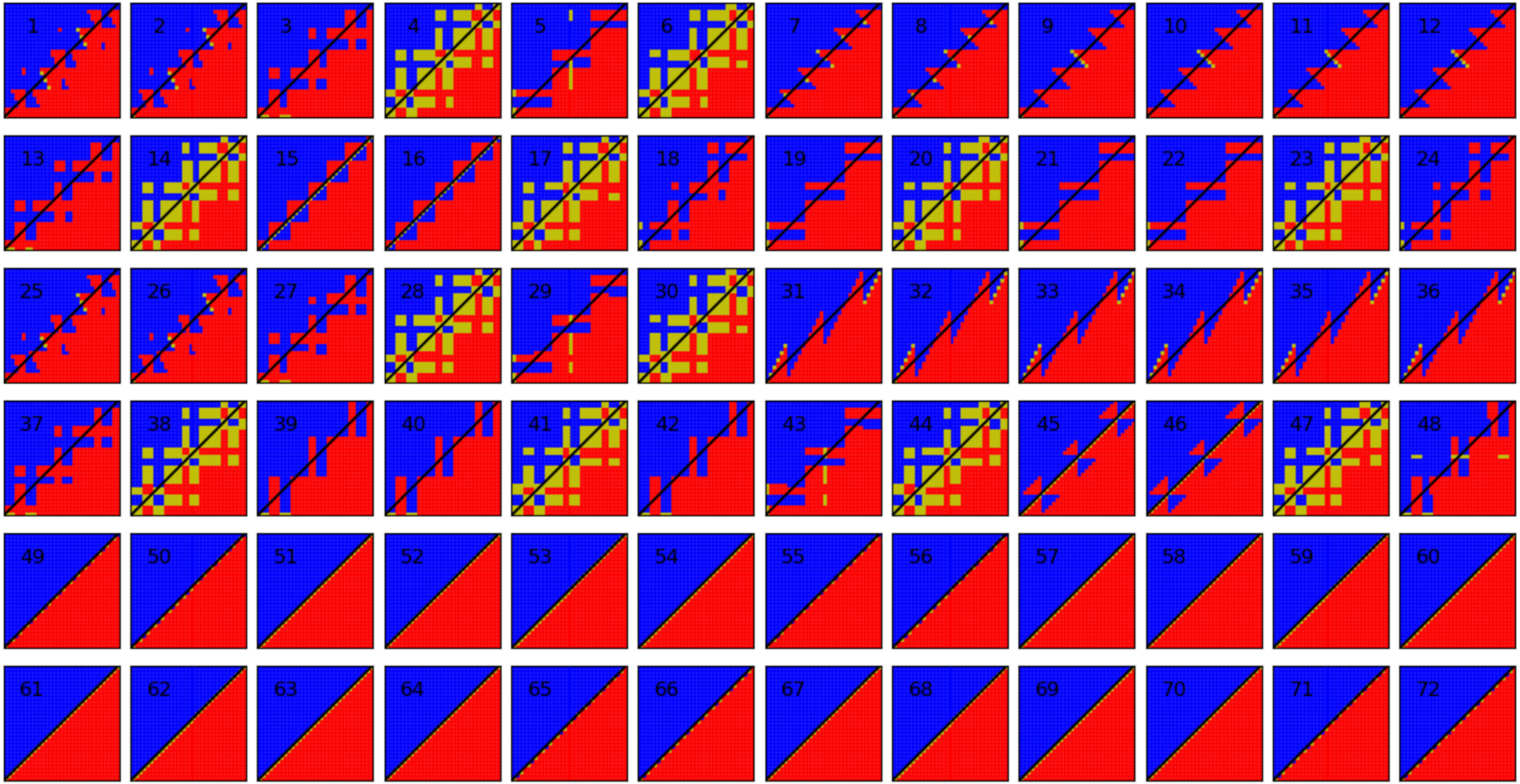
6 ! = 720 different orders to add ⚠

Kettner, Mehlhorn, Pion, Schirra, Yap:
Classroom Examples of Robustness Problems in Geometric Computations

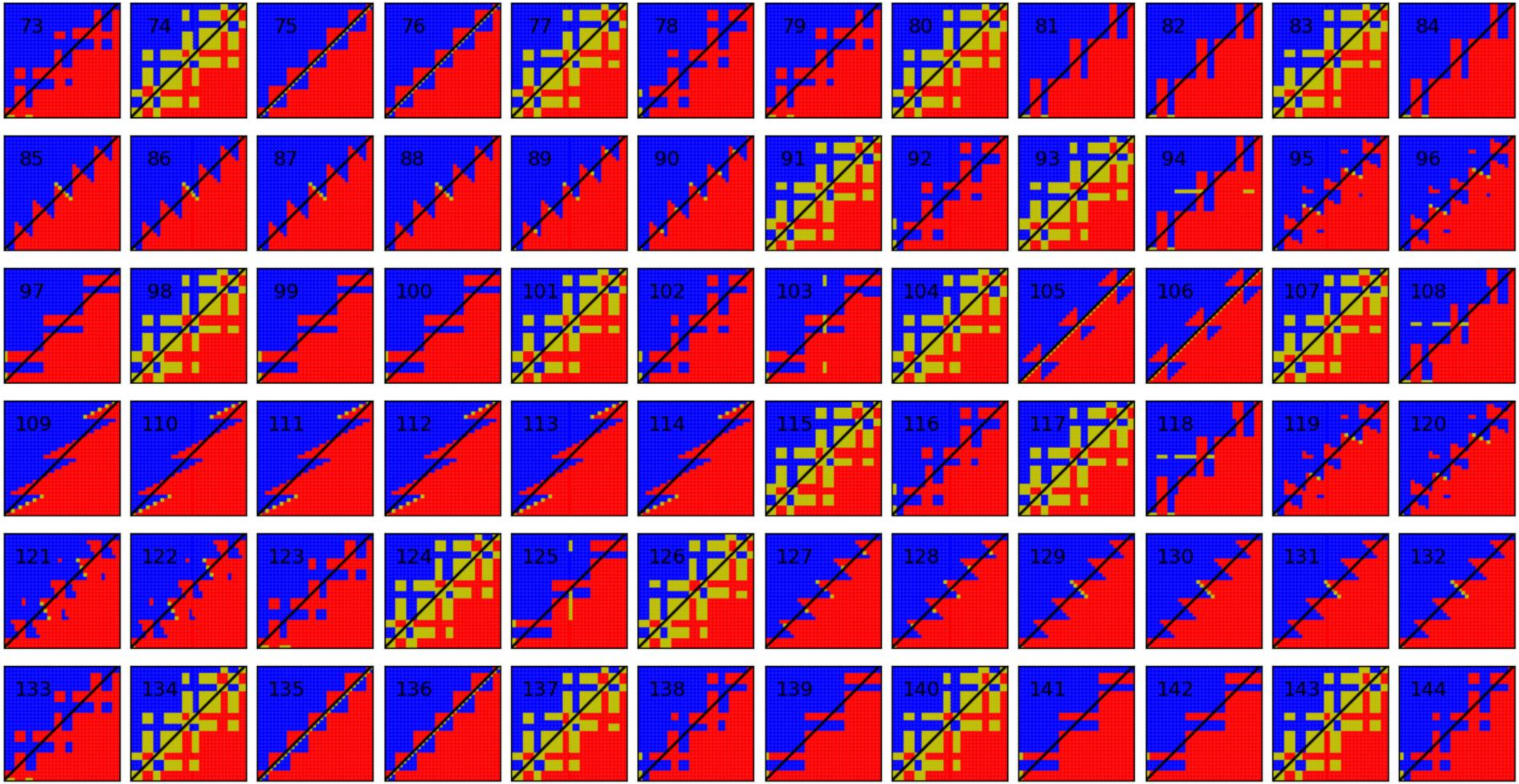
sign-plot.py

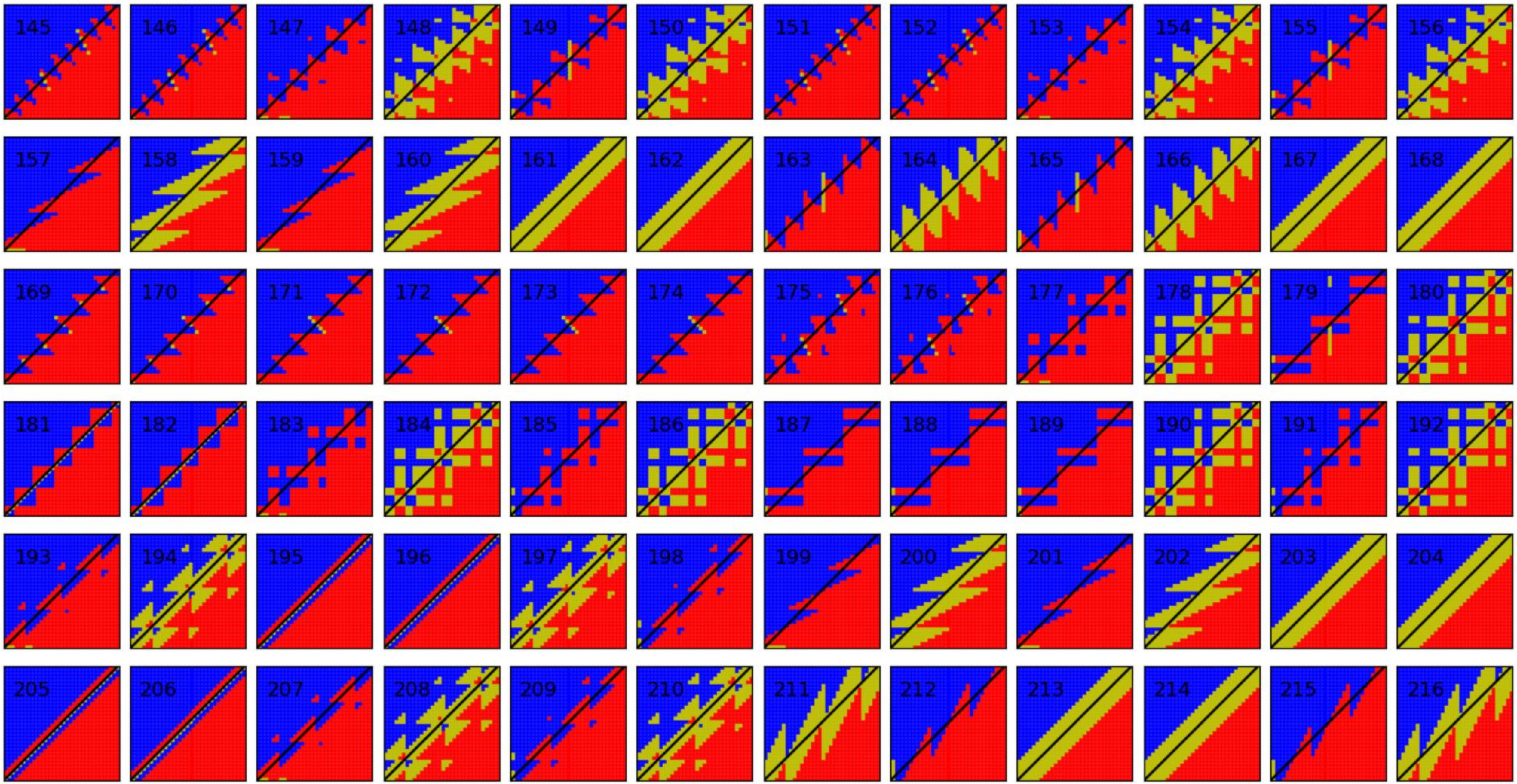
```
import matplotlib.pyplot as plt

N = 256
delta = 1 / 2**54
q = (12, 12)
r = (24, 24)
P = []  # points (i, j, det)

for i in range(N):
    for j in range(N):
        p = (1/2 + i * delta, 1/2 + j * delta)
        det = (q[0]*r[1] + r[0]*p[1] + p[0]*q[1]
             - r[0]*q[1] - p[0]*r[1] - q[0]*p[1])
        P.append((i, j, det))

pos = [(i, j) for i, j, det in P if det > 0]
neg = [(i, j) for i, j, det in P if det < 0]
zero = [(i, j) for i, j, det in P if det == 0]

plt.subplot(facecolor='lightgrey', aspect='equal')
plt.xlabel('i')
plt.ylabel('j', rotation=0)

for points, color in [(pos, "b"), (neg, "r"), (zero, "y")]:
    X = [x for x, y in points]
    Y = [y for x, y in points]
    plt.plot(X, Y, color + ".")

plt.plot([-1, N], [-1, N], "k-")
plt.show()
```

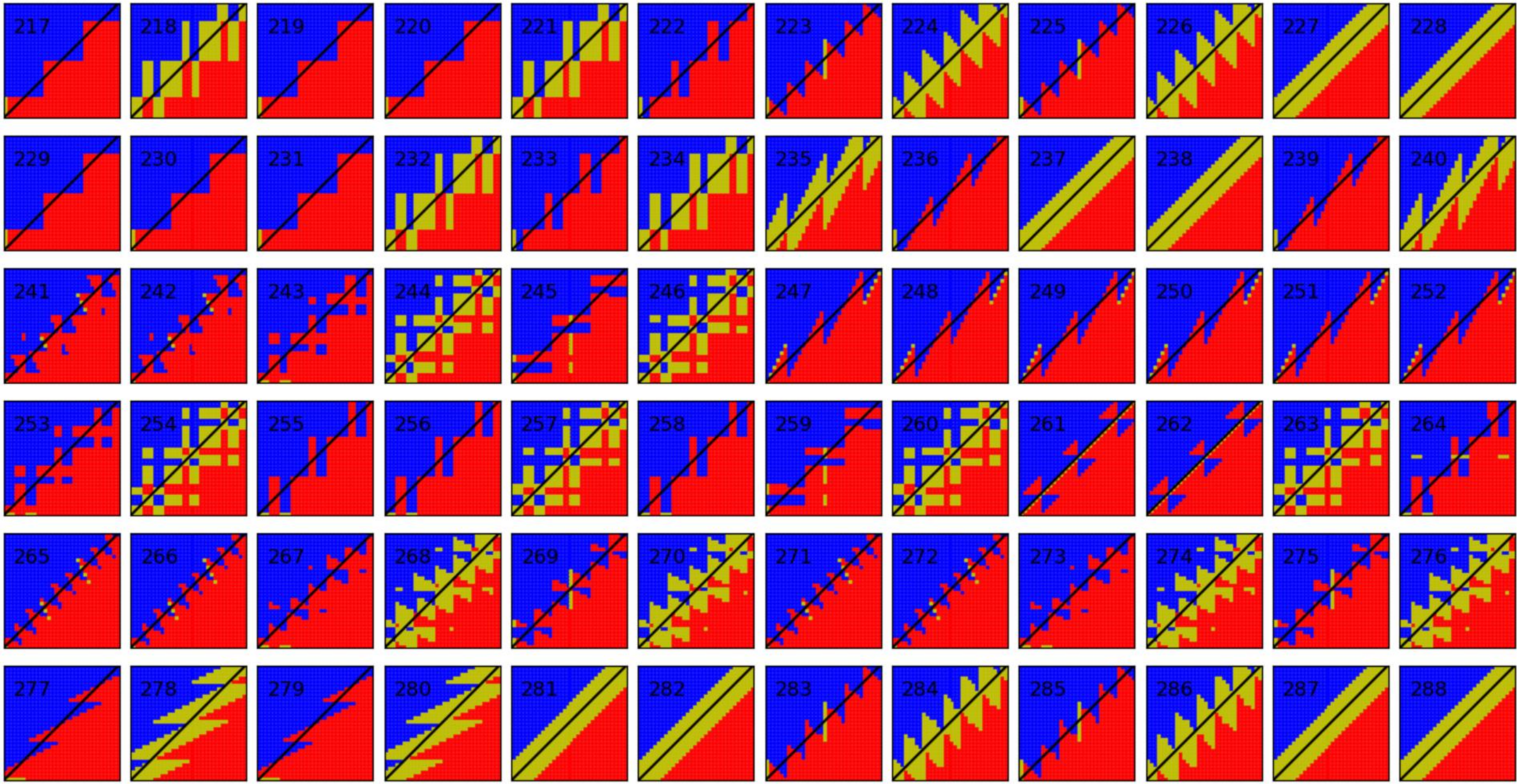


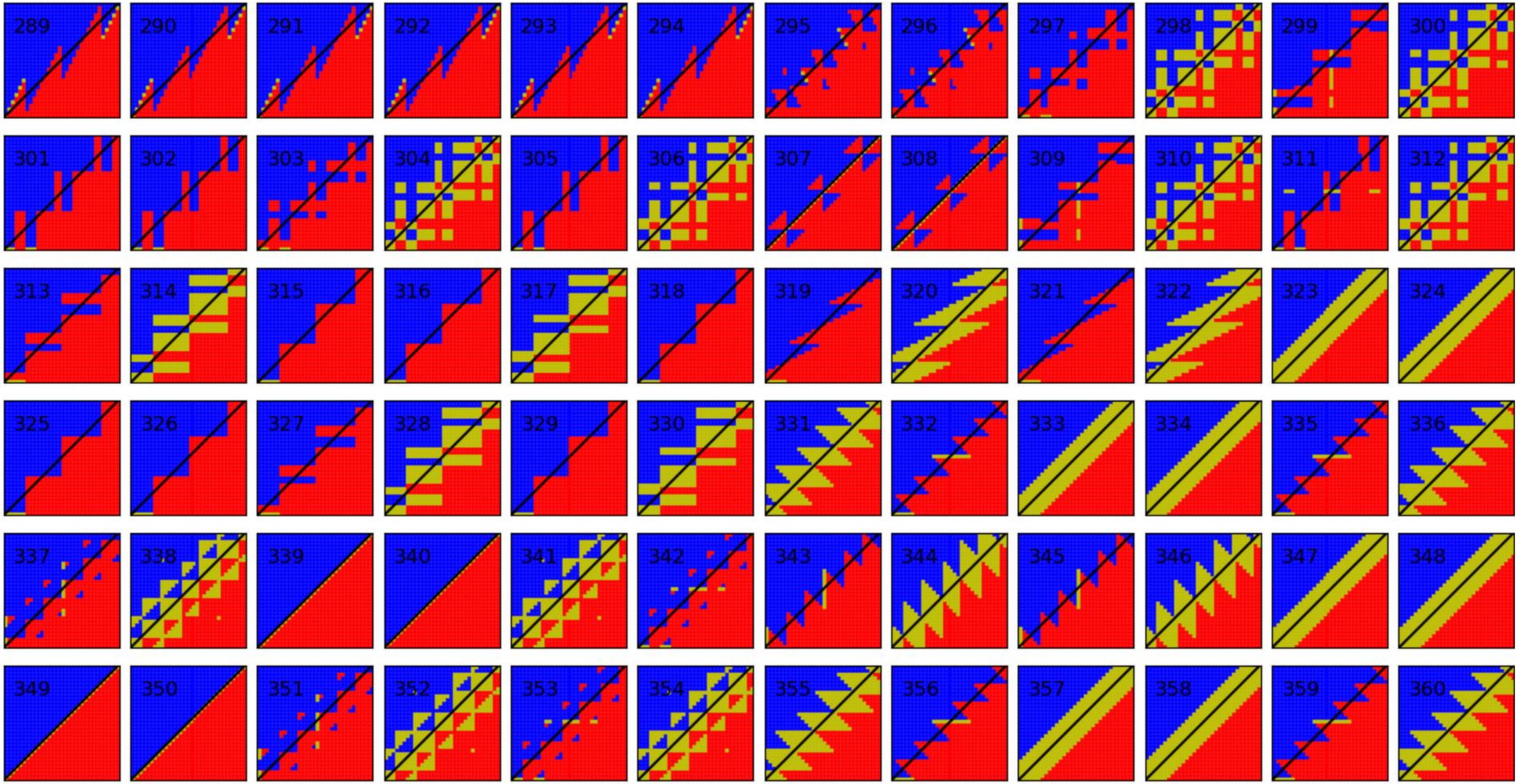


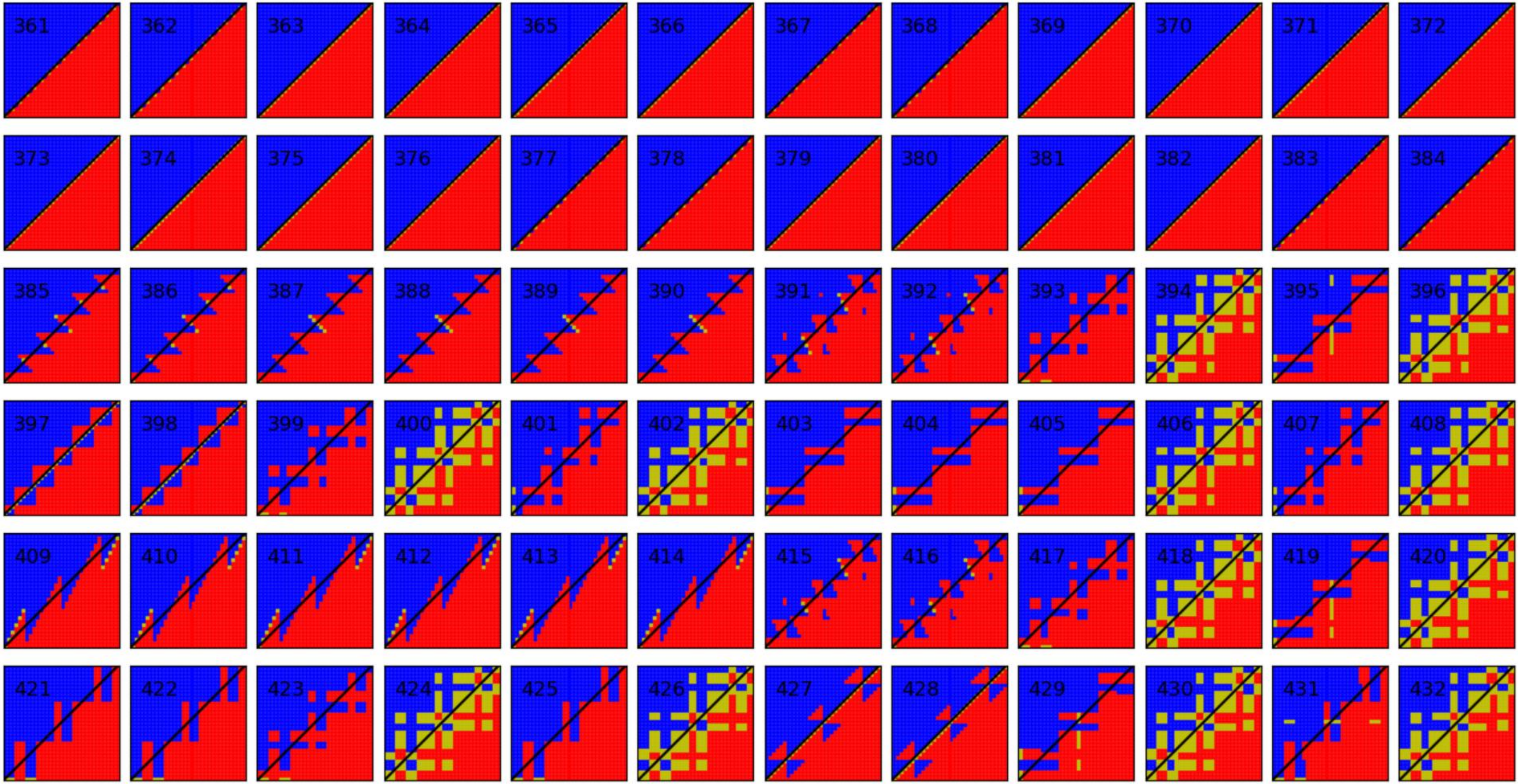
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

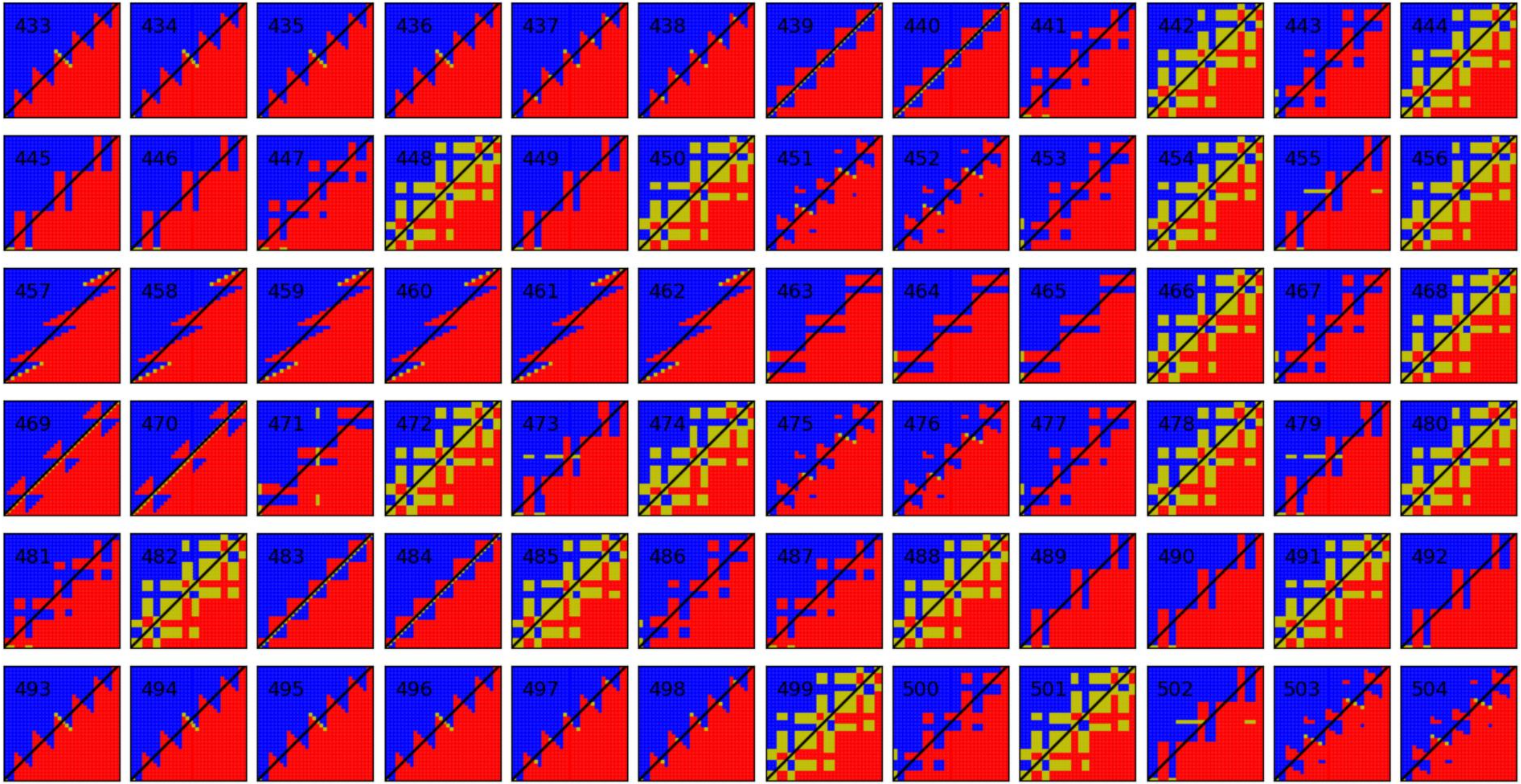

73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144

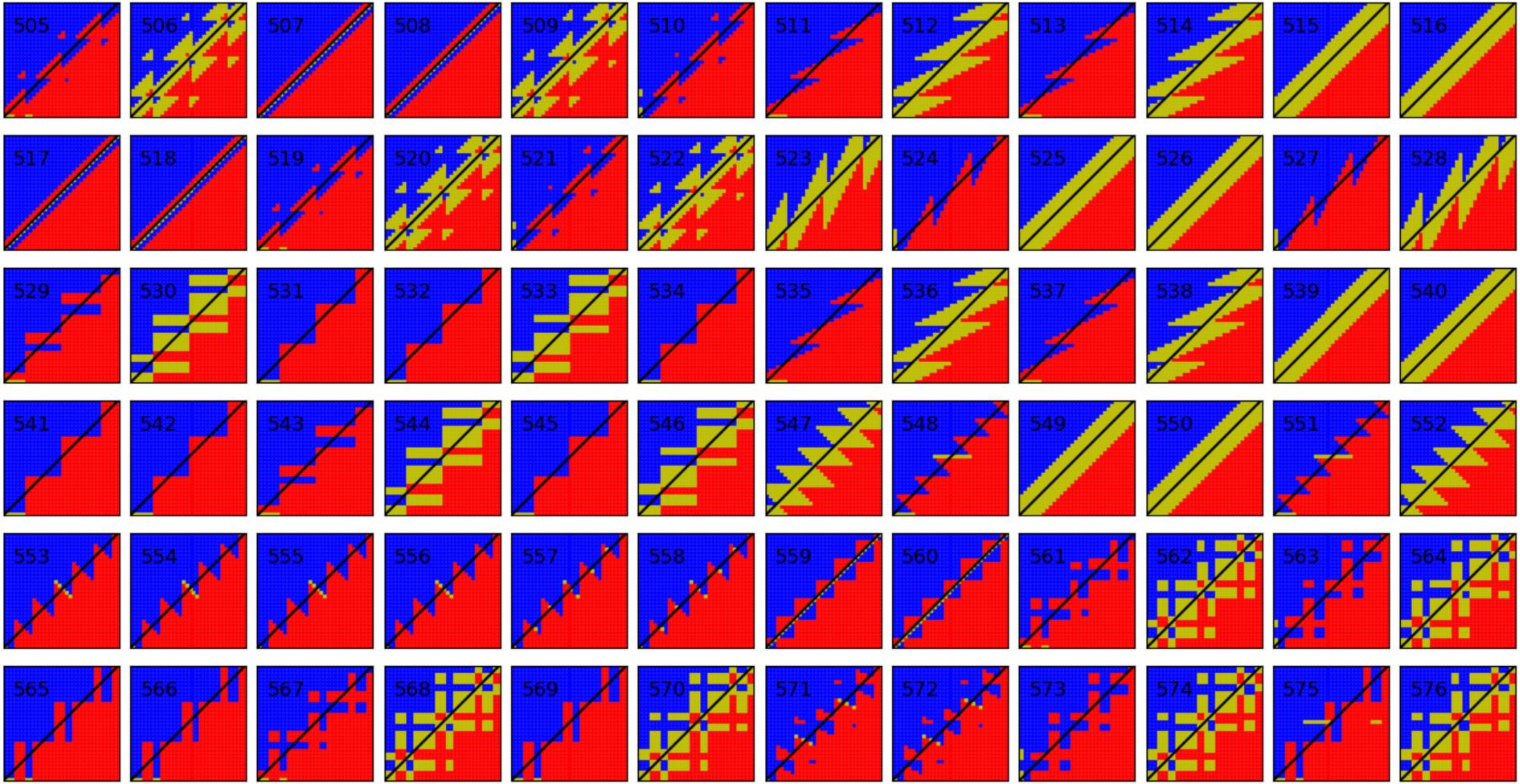

145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216

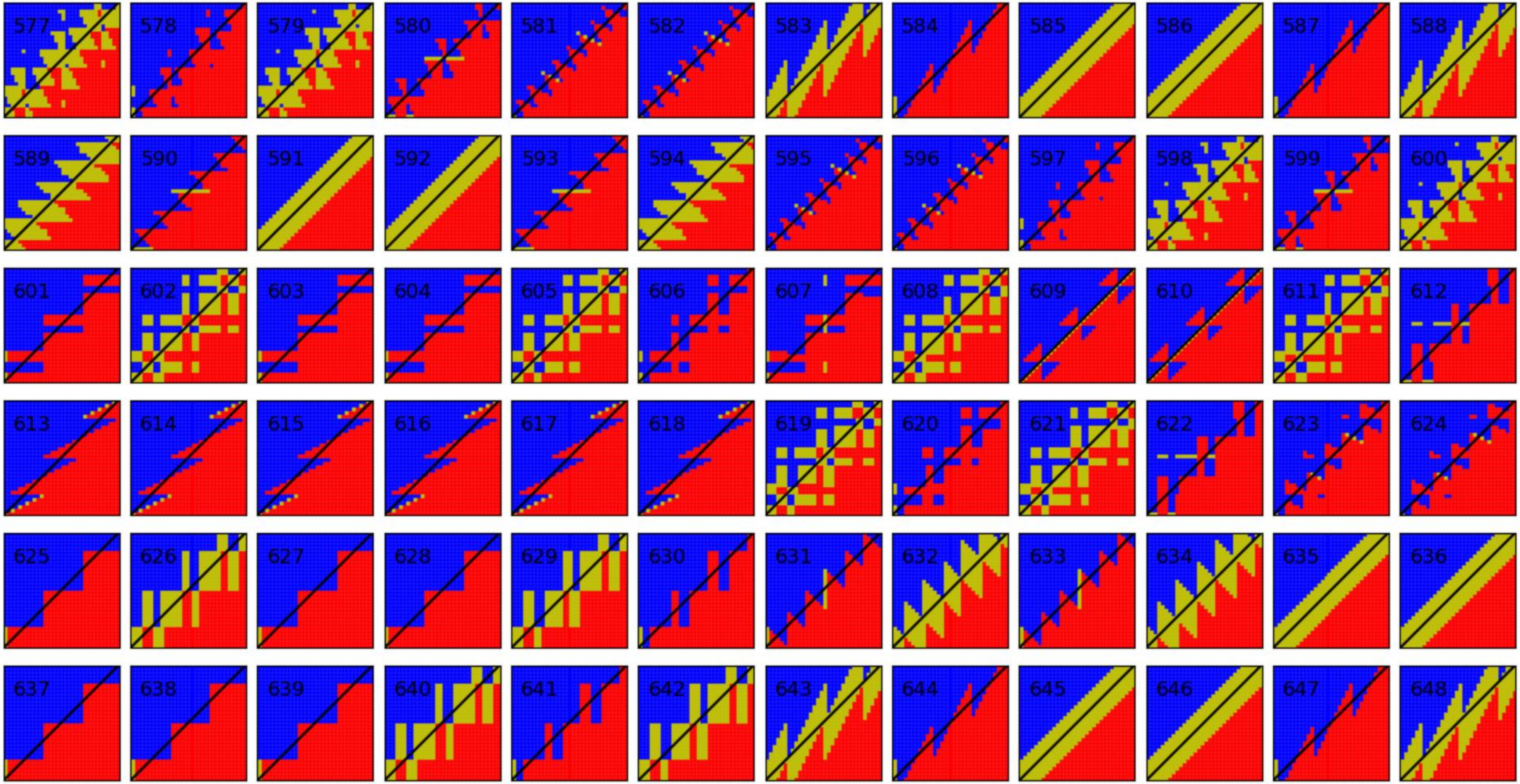

217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288

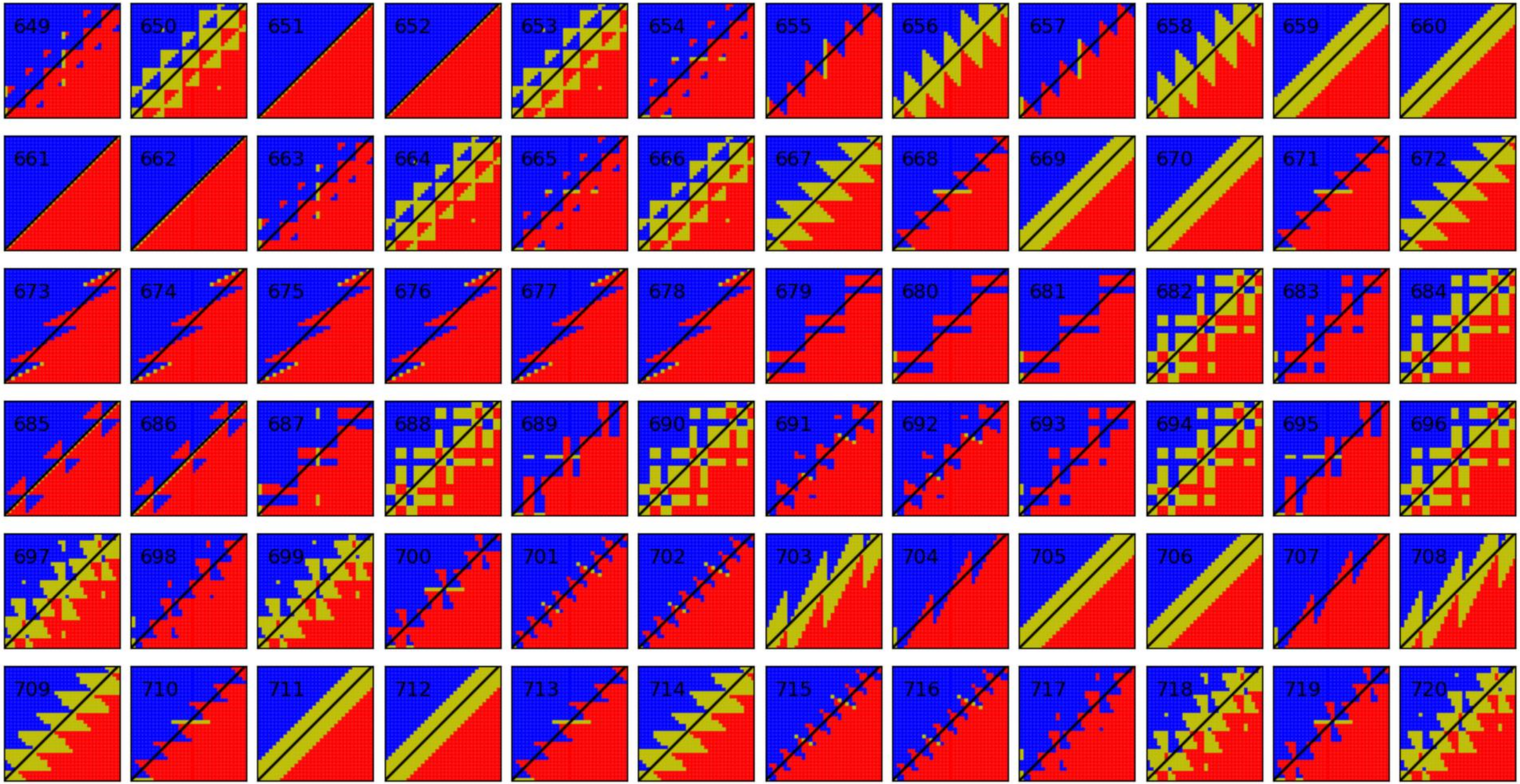

289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360


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