1 Generate Matrix

General method to generate of adjacency matrix.

[8]:
```python
import numpy as np
# to prove the minimal case on not square we need to build matrix for not␣
↪rectangler board
def genenerate_neighbord_matrix_m_n(m,n) -> np.array:
    mat = np.zeros((m*n, m*n), dtype= np.int8)

    # the general case
    for j in range(0, m*n):
        if j-n > -1 :
            mat[j-n,j] = 1

        if j % n != 0 :
            mat[j-1,j] = 1

        mat[j,j] = 1

        if (j+1) % n != 0 :
            mat[j+1,j] = 1

        if j+n < m*n :
            mat[j+n,j] = 1

    return mat
def genenerate_neighbord_matrix(n) -> np.array:
    return genenerate_neighbord_matrix_m_n(n,n)

print('Adj matrix for 3,2 board:')
print(genenerate_neighbord_matrix_m_n(3,2))
```

```
Adj matrix for 3,2 board:
[[1 1 1 0 0 0]
 [1 1 0 1 0 0]
 [1 0 1 1 1 0]
 [0 1 1 1 0 1]
 [0 0 1 0 1 1]
 [0 0 0 1 1 1]]
```

2 Solver based on adjacency matrix

General method to how solve the game, by solving the matrix.

[7]:
```python
from sage.all import *
n = 3
A = Matrix(Integers(2),genenerate_neighbord_matrix(n)) # A = adjacency matrix
```

```
Y = vector([1 for x in range(n**2)]) # Y = ( 1, 1, ..., 1)
X = A.solve_right(Y)
print('Solution for 3x3 board:')
print(X)
```

```
Solution for 3x3 board:
(1, 0, 1, 0, 1, 0, 1, 0, 1)
```

3 Solver based on calculating raw by raw

Second method for solution.

[3]:
```
# row operation on mat to generate the solution for [n, n**2-1]
def gaussian_elimination_spanish_alg(mat : np.array, sol_vec : np.array):
    n = int(sqrt(mat.shape[0]))
    #all rows but the last one
    for i in range(0, n**2-n):
        # the lamp that is affected
        affected_lamp = i + n
        row_i = mat[i][:affected_lamp+1]
        # check rows below
        # for j in range(i+1, n**2):
        for j in [i-1 + n, i+n, i+n+1, i+ 2*n]:
            if j> -1 and j < n**2 and mat[j][affected_lamp] == 1:
                row_j = mat[j][:affected_lamp+1]
                row_j = row_j + row_i
                row_j = row_j % 2
                mat[j][:affected_lamp+1] = row_j
                sol_vec[j] = ( sol_vec[j] + sol_vec[i] ) % 2

# get result to [n, n**2-1] from solution [0, n-1]
def mul_mat_sol_based_on_res(mat : np.array, end_state : list, res : list):
    n = int(sqrt(mat.shape[0]))
    for i in range(0,n**2-n):
        res_i_plus_n = int(end_state[i])
        for j in range(0,i+n):
            res_i_plus_n = (res_i_plus_n + mat[i][j] * res[j]) % 2
        res.append(res_i_plus_n)

# facade for the intire spanish method
def generate_mat_spanish_alg(mat : np.array):
    n = int(sqrt(mat.shape[0]))
    end_state = np.ones(n**2) # end_state = (1, 1, ... , 1)
    gaussian_elimination_spanish_alg(mat, end_state)
    # the matrix we need to solve for parmeter [0, n-1]
    new_mat = np.array(mat[n**2-n:n**2, 0:n], copy=True)
    # the solution vector after row operation
```

```
    new_sol = np.array(end_state[n**2-n:n**2],  copy=True)

    # find solution for n variables
    A = Matrix(Integers(2),new_mat)
    Y = vector(Integers(2),new_sol)
    X = A.solve_right(Y)
    res = [x for x in X] # solution for parmeter [0, n-1]
    mul_mat_sol_based_on_res(mat, end_state, res)
    return res

mat = genenerate_neighbord_matrix(4)
A = Matrix(Integers(2),mat)
res = generate_mat_spanish_alg(mat)
print('solution for board n=4:')
print(res)

print('check solution by multiply matrix with soultion vector:')
X = vector(Integers(2),res)
Y = A*X
print(Y)
```

```
solution for board n=4:
[0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1]
check solution by multiply matrix with soultion vector:
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```

4 Benchmark

Comparing time takes to generate solution based on two solution

[4]:
```
import datetime
import numpy as np

def matrix_solve(mat):
    A = Matrix(Integers(2),mat)
    Y = vector([1 for x in range(n**2)])
    Z = vector([0 for x in range(n**2)])
    X = A.solve_right(Y)
    return X

val = []
# run on range(10 ,61,5)
for i,n in enumerate(range(10 ,15)):
    # print(i)
    mat = genenerate_neighbord_matrix(n)

    a0 = datetime.datetime.now()
```

```
    matrix_solve(mat)
    b0 = datetime.datetime.now()
    c0 = b0 - a0
    t0 = c0.total_seconds()
    # print(t0)

    a1 = datetime.datetime.now()
    generate_mat_spanish_alg(mat)
    b1 = datetime.datetime.now()
    c1 = b1 - a1
    t1 = c1.total_seconds()
    # print(t1)

    val.append((n, t0, t1))

res = np.array(val)
# np.savetxt("benchmark.csv", res, delimiter = ',')
print('board size, adj method, row by row method')
print(res)
```

```
board size, adj method, row by row method
[[10.         0.029358  0.319221]
 [11.         0.042352  0.406416]
 [12.         0.051597  0.548713]
 [13.         0.064825  0.781002]
 [14.         0.101306  1.072234]]
```

5 Optimal solution

searching for integer system solution.

[5]:
```
from sage.all import *
n = 3
m = 2
a = genenerate_neighbord_matrix_m_n(m,n)
A = Matrix(ZZ,a)
Y = vector([1 for x in range(m*n)])
Z = vector([0 for x in range(m*n)])
X = A.solve_right(Y)
print('Optimal solution:')
print(X)
```

```
Optimal solution:
(0, 0, 1, 1, 0, 0)
```

6 Solution Amount

Calculating number of solution based on board size.

[6]:
```
def num_solution_board(m,n):
    a = genenerate_neighbord_matrix_m_n(m, n)
    A = Matrix(Integers(2),a)
    num_solutions = 2**A.kernel().dimension()
    return num_solutions

m = 9
n = 9
res = np.zeros((m, n), dtype= np.int32)
for i in range(1,m+1):
    for j in range(1,n+1):
        res[i-1][j-1] = num_solution_board(i,j)
print('Number solution based on m x n board size:')
print(res)
```

```
Number solution based on m x n board size:
[[  1   2   1   1   2   1   1   2   1]
 [  2   1   4   1   2   1   4   1   2]
 [  1   4   1   1   8   1   1   4   1]
 [  1   1   1  16   1   1   1   1  16]
 [  2   2   8   1   4   1  16   2   2]
 [  1   1   1   1   1   1   1  64   1]
 [  1   4   1   1  16   1   1   4   1]
 [  2   1   4   1   2  64   4   1   2]
 [  1   2   1  16   2   1   1   2 256]]
```