Evals_of_PM_scheme_via_Srinivasan_algorithm

October 7, 2025

[1]:
```
# This SageMath implementation is based on the recursive algorithm described by␣
 ↪Srinivasan in Section 5 of:
#     M. K. Srinivasan, "The perfect matching association scheme",
#     Algebraic Combinatorics 3 (2020), 559-591.
#
# The original Maple implementation of this algorithm, by Srinivasan, is␣
 ↪available in the online notes:
#     M. K. Srinivasan, "A maple program for computing␣
 ↪$\hat{\theta}_{2\mu}^{2\lambda}$"
#     (https://www.math.iitb.ac.in/~mks/papers/EigenMatch.pdf)
#
# The code below translates Srinivasan's Maple program into SageMath syntax to␣
 ↪compute the tables of
# eigenvalues for the graphs in the perfect matching association scheme.
#
# Later on we used the above implementation to generate matrices with␣
 ↪eigenvalues of the perfect matching
# scheme. Then use those matrices to verify our conjectures from the manuscript:
#     H. Gupta, A. Herman, A. Lacaze-Masmonteil, R. Maleki, and K. Meagher,
#     "On the second largest eigenvalue of certain graphs in the perfect␣
 ↪matching
#      association scheme", (2025+),
# about the second-largest eigenvalue and spectral gap of certain graphs
#
# At the end, we also use it to find symmetric functions to compute eigenvalues␣
 ↪for two particular
# columns (also based on a result of Srinivasan).
#
# For more details see the above references.
```

[2]:
```
from sage.all import *
from sage.misc.cachefunc import cached_function
```

[3]:
```
# The following translates Srinivasan's Maple program into SageMath syntax in␣
 ↪the same order.
# For description we refer to https://www.math.iitb.ac.in/~mks/papers/
 ↪EigenMatch.pdf
```

```python
# (1)
@cached_function
def park(n, k):
    if n <= 0 or k <= 0 or k > n:
        return 0
    if n == 1:
        return 1
    return park(n-1, k-1) + park(n-k, k)

# (2)
@cached_function
def par(n):
    """Number of partitions of n."""
    return sum(park(n, k) for k in range(1, n+1))

# (3)
@cached_function
def rankpar(L_tuple):
    """
    Rank of partition L in lexicographic order (1-based).
    Input: tuple of weakly decreasing integers.
    """
    L = list(L_tuple)
    n = sum(L)
    if L[0] == 1:
        return 1
    if len(L) == 1:
        return par(n)
    return sum(park(n, i) for i in range(1, L[0])) + rankpar(tuple(L[1:]))

# (4)
@cached_function
def unrankpar(r, n):
    """
    Partition of n with given rank r (1-based).
    Output as list.
    """
    if r == 1:
        return [1] * n
    j, t = 1, park(n, 1)
    while t + park(n, j+1) < r:
        j += 1
        t += park(n, j)
    return [j+1] + unrankpar(r - t, n - j - 1)

# (5)
```

```python
@cached_function
def ppar(n):
    """Number of pointed partitions of n."""
    if n == 1:
        return 1
    return 1 + sum(par(n - i) for i in range(1, n))

# (6)
@cached_function
def rankppar(L_tuple):
    """
    Rank of pointed partition L (1-based).
    Input: tuple (last element distinguished).
    """
    L = list(L_tuple)
    n = sum(L)
    if len(L) == 1:
        return ppar(n)
    return sum(par(n - i) for i in range(1, L[-1])) + rankpar(tuple(L[:-1]))

# (7)
@cached_function
def unrankppar(r, n):
    """
    Given rank r, return pointed partition of n.
    """
    if r == 1:
        return [1]*n
    if r == ppar(n):
        return [n]
    j, t = 1, 0
    while t + par(n - j) < r:
        t += par(n - j)
        j += 1
    return unrankpar(r - t, n - j) + [j]

# (8)
@cached_function
def cob(L_tuple):
    """
    Given list of row lengths of a Young diagram,
    return contents of outer boxes as strictly decreasing list.
    """
    L = list(L_tuple)
    if len(L) == 1:
        return [L[0], -1]
    S = [x - 1 for x in cob(tuple(L[1:]))]
```

```
    if L[0] == L[1]:
        return [L[0]] + S[1:]
    return [L[0]] + S

# (9)
def insert(L, x):
    """Insert x into weakly decreasing list L."""
    if not L:
        return [x]
    if x >= L[0]:
        return [x] + L
    return [L[0]] + insert(L[1:], x)

# (10)
def updatematch(L, n, a):
    """
    Update vector for (X_{2n-1} - a*I) * v.
    """
    A = [0]*ppar(n)
    for i in range(1, ppar(n)+1):
        S = unrankppar(i, n)

        # merge with pointed part
        for j in range(len(S)-1):
            temp = S[:-1]
            new_list = temp[:j] + temp[j+1:]
            new_partition = new_list + [S[j] + S[-1]]
            u = rankppar(tuple(new_partition))
            A[u-1] += 2 * S[j] * L[i-1]

        k = S[-1]
        # split pointed part
        for j in range(1, k):
            temp = S[:-1]
            new_S = insert(temp, k-j)
            new_S.append(j)
            u = rankppar(tuple(new_S))
            A[u-1] += L[i-1]
            A[i-1] += L[i-1]

    # subtract a*L
    for i in range(ppar(n)):
        A[i] -= a*L[i]
    return A

# (11)
@cached_function
```

```
def thetarow(L_tuple):
    """Compute theta-row for partition L."""
    L = list(L_tuple)
    n = sum(L)//2
    if n == 1:
        return [1]

    # build J
    if L[-1] == 2:
        J = L[:-1]
    else:
        J = L[:-1] + [L[-1]-2]

    S = thetarow(tuple(J))

    R = [0]*ppar(n)
    for i in range(par(n-1)):
        R[i] = S[i]

    F = cob(tuple(J))
    if L[-1] == 2:
        F = F[:-1]
    else:
        F = F[:-2] + F[-1:]

    for f in F:
        R = updatematch(R, n, f)

    T = [0]*par(n)
    for i in range(1, ppar(n)+1):
        u = unrankppar(i, n)
        v = insert(u[:-1], u[-1])
        T[rankpar(tuple(v))-1] += R[i-1]

    m = T[0]
    if m == 0:
        raise ValueError(f"Normalization failed: L={L}, T={T}")
    T = [t//m for t in T]
    return T
```

[4]:
```
# We first define the set of all even partitions and then using that define a␣
 ↪function
# to generate the eigenvalues as matrix whose rows are in the decreasing order␣
 ↪from top to bottom
# while columns are in the increasing order from left to right.

def even_partitions_2n(n):
```

```
    """
    Return all partitions of 2n with all parts even, as lists.
    """
    return [ [2*x for x in p] for p in Partitions(n) ]

def thetahat_matrix(n):
    """
    Construct the matrix whose rows and columns
    are indexed by even_partitions_2n(n).
    """
    ev_parts = even_partitions_2n(n)
    P1 = [thetarow(tuple(p)) for p in ev_parts]
    P = matrix(ZZ, P1)
    return P

def thetahat_all(N):
    """
    Compute thetahat_matrix(n) for all 2 <= n <= N
    Returns dictionary {n: (list_rows, matrix)}.
    """
    results = {}
    for n in range(2, N+1):
        results[n] = thetahat_matrix(n)
    return results
```

[5]:
```
# In our paper, we conjecture that if the graph in the perfect matching␣
 ↪association scheme is
# indexed by partitions containing at least two parts of size 2, then its
# second largest eigenvalue occurs on the  [2n-2,2]-eigenspace. The following␣
 ↪implementation is
# to vertify this conjecture by using the above defined functions.

def main_conjecture(N):
    M = thetahat_all(N)
    conjecture_true = True

    for n in [2..N]:
        even_parts = even_partitions_2n(n)
        A = M[n]
        B = A.transpose()
        s = A.nrows()
        for i in [0..s-2]:
            # Only check partitions with at least two parts equal to 2.
            if even_parts[i].count(2)>=2:
                j = s-1-i # Since rows of B are in the increasing order.
                col = B[j]
                # Find the maximum among all except the first (index =0).
```

```
                m = max(col[l] for l in [1..(s-1)])
                t = list(col).index(m)
                # If you wish to output the numbers, ADD this ->␣
↪print(n,j,even_parts[i],col,t)
                # If the maximum does not occur at index 1 (corresponding to␣
↪[2n-2,2]).
                if t != 1:
                    print(f" Conjecture is FALSE for n = {n}, partition =␣
↪{even_parts[i]}")
                    conjecture_true = False
                    return # Stop immediately if a counterexample is found.
    if conjecture_true:
        print("Second largest eigenvalue conjecture is True for all tested n␣
↪ ", N)
```

[6]:
```
# In our paper, we conjecture that among the non-identity adjacencey matrices␣
↪of the
# perfect matching scheme, the minimum spectral gap is attained by␣
↪A_{[2,1^{n-2}]} provided
# n \neq 4. The following is a function to verify this conjecture.

def conjecture_min_spectral_gap(N):
    M = thetahat_all(N)
    conjecture_true = True

    for n in [2..N]:
        A = M[n]
        B = A.transpose()
        s = A.nrows()
        list_for_spectral_gaps = []
        for i in [1..s-1]:
            col = list(B[i])
            m = max(col[l] for l in [1..(s-1)])
            spectral_gap = col[0]-m
            list_for_spectral_gaps = list_for_spectral_gaps + [spectral_gap]
        min_spec_gap = min(list_for_spectral_gaps)
        partition_for_min_spectral_gap = list_for_spectral_gaps.
↪index(min_spec_gap)
        if n!=4 and partition_for_min_spectral_gap != 0:
            print(f" Conjecture is FALSE for n = {n}")
            conjecture_true = False
            return # Stop immediately if a counterexample is found.
    if conjecture_true:
        print("Spectral gap conjecture is True for all tested n  ", N)
```

[7]:
```
# The following function takes input as n and power k. It runs over all the␣
 ↪even partitions of 2n
# and outputs corresponding sum of powers of contents that partition as a list.

def content_even_partitions(n, k):
    parts = even_partitions_2n(n)
    values = []
    for lam in parts:
        total = 0
        for i, row_length in enumerate(lam):
            for j in range(row_length):
                c = j - i
                total += c^k
        values.append(total)
    return values
```

[8]:
```
# Choose a value for N. M will be all the matrices with eigenvalues from
# 2 to N. For example, M[4] will be the matrix with eigenvalues of perfect␣
 ↪matching scheme
# of complete graph K_{8}. Please note that rows are in the decreasing order of␣
 ↪even partitions
# from top to bottom while columns are in the increasing order of even␣
 ↪partitions from left to right.

N = 15
M = thetahat_all(N)
```

[9]:
```
# Let us test the validity of our conjectures.
# For N = 15 it tests within minutes.

N = 15
main_conjecture(N)
conjecture_min_spectral_gap(N)
```

```
Second largest eigenvalue conjecture is True for all tested n   15
Spectral gap conjecture is True for all tested n   15
```

[10]:
```
# In the following function we compute symmetric E_{3,2} in terms of
# elementary power symmetric function where coefficients which are polynomials
# in the variable t is the output. It uses Sage's inbuilt functions␣
 ↪solve_left() to
# solve system of linear equations and for interpolation it uses R.
 ↪langrange_polynomial().

p3_5 = content_even_partitions(5,3)
p2_5 = content_even_partitions(5,2)
p1_5 = content_even_partitions(5,1)
```

```
a5 = len(p1_5)
p0_5 = vector([1 for i in range(a5)])
p1p2_5 = vector([p1_5[i]*p2_5[i] for i in range(a5)])
p1p1_5 = vector([p1_5[i]*p1_5[i] for i in range(a5)])

v5 = matrix(M[5].column(4))
B5 = matrix([p3_5,p1p2_5,p2_5,p1p1_5,p1_5,p0_5])
w5 = B5.solve_left(v5)[0]

p3_6 = content_even_partitions(6,3)
p2_6 = content_even_partitions(6,2)
p1_6 = content_even_partitions(6,1)
a6 = len(p1_6)
p0_6 = vector([1 for i in range(a6)])
p1p2_6 = vector([p1_6[i]*p2_6[i] for i in range(a6)])
p1p1_6 = vector([p1_6[i]*p1_6[i] for i in range(a6)])

v6 = matrix(M[6].column(5))
B6 = matrix([p3_6,p1p2_6,p2_6,p1p1_6,p1_6,p0_6])
w6 = B6.solve_left(v6)[0]

p3_7 = content_even_partitions(7,3)
p2_7 = content_even_partitions(7,2)
p1_7 = content_even_partitions(7,1)
a7 = len(p1_7)
p0_7 = vector([1 for i in range(a7)])
p1p2_7 = vector([p1_7[i]*p2_7[i] for i in range(a7)])
p1p1_7 = vector([p1_7[i]*p1_7[i] for i in range(a7)])

v7 = matrix(M[7].column(5))
B7 = matrix([p3_7,p1p2_7,p2_7,p1p1_7,p1_7,p0_7])
w7 = B7.solve_left(v7)[0]

p3_8 = content_even_partitions(8,3)
p2_8 = content_even_partitions(8,2)
p1_8 = content_even_partitions(8,1)
a8 = len(p1_8)
p0_8 = vector([1 for i in range(a8)])
p1p2_8 = vector([p1_8[i]*p2_8[i] for i in range(a8)])
p1p1_8 = vector([p1_8[i]*p1_8[i] for i in range(a8)])

v8 = matrix(M[8].column(6))
B8 = matrix([p3_8,p1p2_8,p2_8,p1p1_8,p1_8,p0_8])
w8 = B8.solve_left(v8)[0]

z = [[(10,w5[i]),(12,w6[i]),(14,w7[i]),(16,w8[i])] for i in range(6)]
```

```
R.<t> = PolynomialRing(QQ)
poly = [R.lagrange_polynomial(z[i]) for i in range(6)]
show(poly)
```

$$\left[-2, \frac{1}{4}, -\frac{1}{8}t + \frac{15}{2}, -\frac{1}{2}, -\frac{1}{8}t^2 + \frac{29}{8}t - 15, \frac{1}{16}t^3 - \frac{47}{16}t^2 + \frac{29}{4}t\right]$$

[11]:
```
# In the following function we compute symmetric E_{5} in terms of
# elementary power symmetric function where coefficients which are polynomials
# in the variable t is the output. It uses Sage's inbuilt functions␣
 ↪solve_left() to
# solve system of linear equations and for interpolation it uses R.
 ↪langrange_polynomial().

p4_5 = content_even_partitions(5,4)
p3_5 = content_even_partitions(5,3)
p2_5 = content_even_partitions(5,2)
p1_5 = content_even_partitions(5,1)
a5 = len(p1_5)
p0_5 = vector([1 for i in range(a5)])
p1p1_5 = vector([p1_5[i]*p1_5[i] for i in range(a5)])

v5 = matrix(M[5].column(6))
B5 = matrix([p4_5,p3_5,p2_5,p1p1_5,p1_5,p0_5])
w5 = B5.solve_left(v5)[0]

p4_6 = content_even_partitions(6,4)
p3_6 = content_even_partitions(6,3)
p2_6 = content_even_partitions(6,2)
p1_6 = content_even_partitions(6,1)
a6 = len(p1_6)
p0_6 = vector([1 for i in range(a6)])
p1p1_6 = vector([p1_6[i]*p1_6[i] for i in range(a6)])

v6 = matrix(M[6].column(9))
B6 = matrix([p4_6,p3_6,p2_6,p1p1_6,p1_6,p0_6])
w6 = B6.solve_left(v6)[0]

p4_7 = content_even_partitions(7,4)
p3_7 = content_even_partitions(7,3)
p2_7 = content_even_partitions(7,2)
p1_7 = content_even_partitions(7,1)
a7 = len(p1_7)
p0_7 = vector([1 for i in range(a7)])
p1p1_7 = vector([p1_7[i]*p1_7[i] for i in range(a7)])

v7 = matrix(M[7].column(11))
B7 = matrix([p4_7,p3_7,p2_7,p1p1_7,p1_7,p0_7])
```

```
w7 = B7.solve_left(v7)[0]

p4_8 = content_even_partitions(8,4)
p3_8 = content_even_partitions(8,3)
p2_8 = content_even_partitions(8,2)
p1_8 = content_even_partitions(8,1)
a8 = len(p1_8)
p0_8 = vector([1 for i in range(a8)])
p1p1_8 = vector([p1_8[i]*p1_8[i] for i in range(a8)])

v8 = matrix(M[8].column(15))
B8 = matrix([p4_8,p3_8,p2_8,p1p1_8,p1_8,p0_8])
w8 = B8.solve_left(v8)[0]

z = [[(10,w5[i]),(12,w6[i]),(14,w7[i]),(16,w8[i])] for i in range(6)]

R.<t> = PolynomialRing(QQ)
poly = [R.lagrange_polynomial(z[i]) for i in range(6)]
show(poly)
```

$$\left[\frac{1}{2}, -4, -\frac{3}{2}t + 20, -1, 7t - 34, \frac{5}{12}t^3 - 8t^2 + \frac{217}{12}t\right]$$

```
[ ]:
```