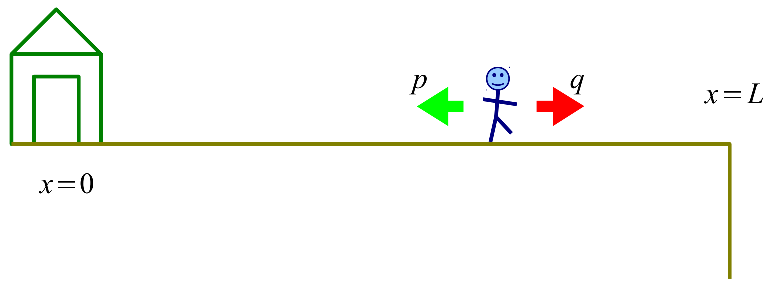
矩陣的本徵問題



移轉矩陣：隨機漫步

$s(x)$：回家機率

$\tau(x)$：平均回家時間

$$s(x) = p\, s(x-1) + (1-p)\, s(x+1)$$

$$\begin{pmatrix} s(x+1) \\ s(x) \end{pmatrix} = \begin{pmatrix} \frac{1}{1-p} & \frac{-p}{1-p} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} s(x) \\ s(x-1) \end{pmatrix}$$

$$\tau(x) = \frac{p\,s(x-1)}{s(x)}\tau(x-1) + \frac{(1-p)\,s(x+1)}{s(x)}\tau(x+1) + 1$$

$$g(x) \equiv \tau(x)\, s(x) \qquad g(x) = p\,g(x-1) + (1-p)\,g(x+1) + s(x)$$

$$\begin{pmatrix} g(x+1) \\ g(x) \\ s(x+1) \\ s(x) \end{pmatrix} = \begin{pmatrix} \frac{1}{1-p} & \frac{-p}{1-p} & \frac{-1}{1-p} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{1-p} & \frac{-p}{1-p} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} g(x) \\ g(x-1) \\ s(x) \\ s(x-1) \end{pmatrix}$$

$p = 1/2$ 時

$s(x) = 1 - x/L$

$\tau(x) = 2\,L\,x/3 - x^2/3$

大矩陣案例：基因突變

DNA sequence

$$\begin{tikzcd} \text{ATGCCCGGAATTTGCT} \arrow[r] & 1100000011111001 \arrow[d] \\ & i \end{tikzcd}$$

$$q_{i,j} = u^{h_{i,j}}(1-u)^{L-h_{i,j}}$$

$h_{i,j}$: Hamming distance between i and j.

$$X_i(t+1) = \sum_j q_{i,j} f_j X_j(t)$$

Quasi-species equation

Single Peak fitness landscape: $F_{i,j} = (f_A \delta_{i,0} + 1)\delta_{i,j}$

Transfer matrix: $\mathbf{T} = \mathbf{q} \cdot \mathbf{F}$

This is a full matrix.

The stationary state of the system corresponds to the eigenvector of the matrix with the largest eigenvalue.

大矩陣案例：　Schrödinger 方程

$$\nabla^2\phi+\frac{2m}{\hbar}[E-V(x)]\phi=0$$

$$E\phi=-\nabla^2\phi+V\phi$$

$\boxed{\text{二維}, 2m=\hbar}$

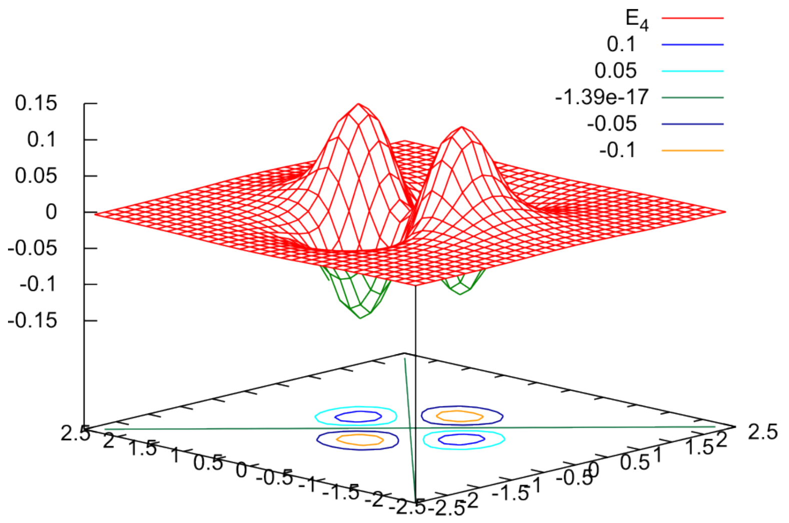
$$\simeq\frac{1}{a^2}\left(\phi_L+\phi_R+\phi_U+\phi_D-4\phi\right)+V(x,y)\phi$$

a: lattice spacing

$\boldsymbol{\phi}_L=\mathbf{T}_L\cdot\boldsymbol{\phi}$　　$T_{L\,i,j}=1$ if j is the left neighbor of i, 0 otherwise

On a 32×32 lattice, we need to find the eigensystem of a 1024×1024 matrix.

This is a sparse matrix.



本徵問題的數值解

- Power iteration
- QR algorithm

see http://en.wikipedia.org/wiki/List_of_numerical_analysis_topics#Eigenvalue_algorithms

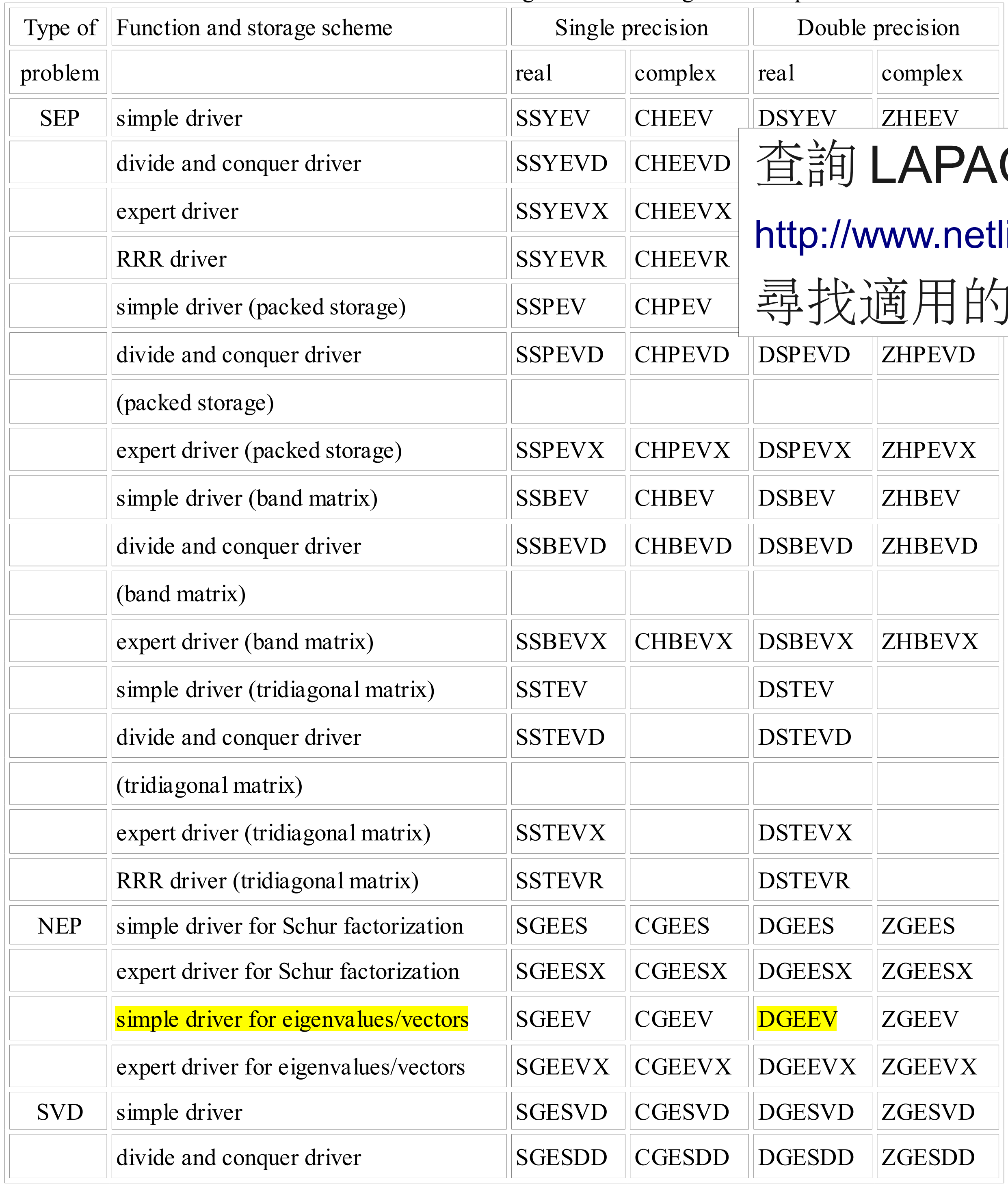
線性代數程式庫:

- LAPACK: for dense matrix
- ARPACK: sparse matrix

實例： LAPACK 使用

- 決定使用軟體
- 閱讀使用手冊及使用範例
- 對照參考手冊，了解輸入要求及輸出方式
- 寫簡單的程式來驗證所得到的了解
- 運用到要所需要的計算上
- [Optional] 效能驗證 (benchmark)

Table 2.5: Driver routines for standard eigenvalue and singular value problems



Type of	Function and storage scheme	Single precision		Double precision	
problem		real	complex	real	complex
SEP	simple driver	SSYEV	CHEEV	DSYEV	ZHEEV
	divide and conquer driver	SSYEVD	CHEEVD	[illegible]	[illegible]
	expert driver	SSYEVX	CHEEVX	[illegible]	[illegible]
	RRR driver	SSYEVR	CHEEVR	[illegible]	[illegible]
	simple driver (packed storage)	SSPEV	CHPEV	[illegible]	[illegible]
	divide and conquer driver	SSPEVD	CHPEVD	DSPEVD	ZHPEVD
	(packed storage)				
	expert driver (packed storage)	SSPEVX	CHPEVX	DSPEVX	ZHPEVX
	simple driver (band matrix)	SSBEV	CHBEV	DSBEV	ZHBEV
	divide and conquer driver	SSBEVD	CHBEVD	DSBEVD	ZHBEVD
	(band matrix)				
	expert driver (band matrix)	SSBEVX	CHBEVX	DSBEVX	ZHBEVX
	simple driver (tridiagonal matrix)	SSTEV		DSTEV	
	divide and conquer driver	SSTEVD		DSTEVD	
	(tridiagonal matrix)				
	expert driver (tridiagonal matrix)	SSTEVX		DSTEVX	
	RRR driver (tridiagonal matrix)	SSTEVR		DSTEVR	
NEP	simple driver for Schur factorization	SGEES	CGEES	DGEES	ZGEES
	expert driver for Schur factorization	SGEESX	CGEESX	DGEESX	ZGEESX
	simple driver for eigenvalues/vectors	SGEEV	CGEEV	DGEEV	ZGEEV
	expert driver for eigenvalues/vectors	SGEEVX	CGEEVX	DGEEVX	ZGEEVX
SVD	simple driver	SGESVD	CGESVD	DGESVD	ZGESVD
	divide and conquer driver	SGESDD	CGESDD	DGESDD	ZGESDD

查詢 LAPACK Users' Guide

http://www.netlib.org/lapack/lug/node32.html

尋找適用的函數

使用 Fortran 程式庫須注意:

- Fortran 的參數是 call by reference， 在 C 或 C++ 中須以指標傳送。
- Fortran 的陣列格式索引是從 1 開始。
- 二維陣列的記憶體安排是後索引主導。

DGEEV man page

```
DGEEV(3lapack)       LAPACK driver routine (version 3.3.0)      DGEEV(3lapack)


NAME
       LAPACK-3  -  computes for an N-by-N real nonsymmetric matrix A, the ei-
       genvalues and, optionally, the left and/or right eigenvectors

SYNOPSIS
       SUBROUTINE DGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR,  LDVR,
                         WORK, LWORK, INFO )

           CHARACTER     JOBVL, JOBVR

           INTEGER       INFO, LDA, LDVL, LDVR, LWORK, N

           DOUBLE        PRECISION  A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
                         WI( * ), WORK( * ), WR( * )

PURPOSE
       DGEEV computes for an N-by-N real nonsymmetric matrix A, the  eigenval-
       ues and, optionally, the left and/or right eigenvectors.
        The right eigenvector v(j) of A satisfies
                         A * v(j) = lambda(j) * v(j)
        where lambda(j) is its eigenvalue.
        The left eigenvector u(j) of A satisfies
                      u(j)**H * A = lambda(j) * u(j)**H
        where u(j)**H denotes the conjugate transpose of u(j).
        The computed eigenvectors are normalized to have Euclidean norm
        equal to 1 and largest component real.

ARGUMENTS
        JOBVL   (input) CHARACTER*1
                = 'N': left eigenvectors of A are not computed;
                = 'V': left eigenvectors of A are computed.

        JOBVR   (input) CHARACTER*1
                = 'N': right eigenvectors of A are not computed;
                = 'V': right eigenvectors of A are computed.

        N       (input) INTEGER
                The order of the matrix A. N >= 0.

        A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                On entry, the N-by-N matrix A.
                On exit, A has been overwritten.

        LDA     (input) INTEGER
                The leading dimension of the array A.  LDA >= max(1,N).
```

在 CP1 SSH 伺服器上可用 man dgeev 來得到

```
WR      (output) DOUBLE PRECISION array, dimension (N)
        WI      (output) DOUBLE PRECISION array, dimension (N)
        WR and WI contain the real and imaginary parts,
        respectively, of the computed eigenvalues.  Complex
        conjugate pairs of eigenvalues appear consecutively
        with the eigenvalue having the positive imaginary part
        first.

VL      (output) DOUBLE PRECISION array, dimension (LDVL,N)
        If JOBVL = 'V', the left eigenvectors u(j) are stored one
        after another in the columns of VL, in the same order
        as their eigenvalues.
        If JOBVL = 'N', VL is not referenced.
        If the j-th eigenvalue is real, then u(j) = VL(:,j),
        the j-th column of VL.
        If the j-th and (j+1)-st eigenvalues form a complex
        conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and
        u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL    (input) INTEGER
        The leading dimension of the array VL.  LDVL >= 1; if
        JOBVL = 'V', LDVL >= N.

VR      (output) DOUBLE PRECISION array, dimension (LDVR,N)
        If JOBVR = 'V', the right eigenvectors v(j) are stored one
        after another in the columns of VR, in the same order
        as their eigenvalues.
        If JOBVR = 'N', VR is not referenced.
        If the j-th eigenvalue is real, then v(j) = VR(:,j),
        the j-th column of VR.
        If the j-th and (j+1)-st eigenvalues form a complex
        conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and
        v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR    (input) INTEGER
        The leading dimension of the array VR.  LDVR >= 1; if
        JOBVR = 'V', LDVR >= N.

WORK      (workspace/output)   DOUBLE   PRECISION   array,   dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The dimension of the array WORK.  LWORK >= max(1,3*N), and
        if JOBVL = 'V' or JOBVR = 'V', LWORK >= 4*N.  For good
        performance, LWORK must generally be larger.
        If LWORK = -1, then a workspace query is assumed; the routine
        only calculates the optimal size of the WORK array, returns
        this value as the first entry of the WORK array, and no error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if INFO = i, the QR algorithm failed to compute all the
        eigenvalues, and no eigenvectors have been computed;
        elements i+1:N of WR and WI contain eigenvalues which
        have converged.
```

```cpp
#include <iostream>
#include <cmath>
extern "C" void dgeev_(char * jobvl, char * jobvr, int * n, double * a,
        int * lda, double * wr, double * wi, double * vl, int * ldvl,
        double * vr, int * ldvr, double * work, int * lwork, int * info);
size_t len;
double mrate, peak;
int hamming_dist(size_t i, size_t j)
{
        // count the difference in bits, assume 15 bits maximum
        // using MIT HAKMEM count
        int b = i ^ j;
        size_t k = b - ((b / 2) & 14043) - ((b / 4) & 4681);
        b = ((k + k / 8) & 29127) % 63;
        return b;
}
void set_matrix_eigen(double mat [])
{
        size_t msz = 1u << len; // 2^len
        // mutation matrix
        for (size_t j = 0; j < msz; j ++) for (size_t i = 0; i < msz; i ++) {
                int hd = hamming_dist(i, j);
                mat[j * msz + i] = pow(mrate, hd) * pow(1 - mrate, len - hd);
        }
        // right multipling by the growth matrix
        // and subtracted by the identity
        for (size_t i = 0; i < msz; i ++) {
                mat[i] *= peak;
                mat[i * msz + i] -= 1;
        }
}
```

```cpp
int main(int argc, char ** argv)
{
        len = 8; mrate = 0.1; peak = 2;
        size_t msz = 1 << len;
        double ev[msz], gr[msz];
        mutmat(ev, gr);
        double norm = 0;
        for (size_t i = 0; i < msz; i ++) norm += gr[i];
        std::cout << "peak=" << gr[0] / norm << '\n';
        for (size_t i = 0; i < 8; i ++) std::cout << "ev" << i << '=' << ev[i] << '\n';
        return 0;
}
```

```cpp
void mutmat(double eigenv [], double ground [])
{
        int msz = 1u << len; // 2^len
        double * mat = new double [msz * msz];
        double * wr = new double [msz];
        double * wi = new double [msz];
        double * vr = new double [msz * msz];
        int * evo = new int [msz];
        int lwork = 64 * msz;
        double * work = new double [lwork];
        set_matrix_eigen(mat);
        char jobvl = 'N', jobvr = 'V';
        double dummy;
        int info, ldvl = 1;
        dgeev_(& jobvl, & jobvr, & msz, mat, & msz, wr, wi,
                & dummy, & ldvl, vr, & msz, work, & lwork,
                & info);
        if (info) {
                std::cerr << "info=" << info << "lwork=" << lwork
                        << ", optionallwork=" << work[0] << '\n';
                return;
        }
        // sort the eigenvalues
        for (int i = 0; i < msz; i ++) evo[i] = i;
        for (int i = 0; i < msz; i ++) for (int j = i + 1; j < msz; j ++) {
                if (wr[evo[j]] > wr[evo[i]]) {
                        int k = evo[j];
                        evo[j] = evo[i];
                        evo[i] = k;
                }
        }
        for (int i = 0; i < msz; i ++) {
                eigenv[i] = wr[evo[i]];
                ground[i] = vr[evo[0] * msz + i];
        }
        delete [] mat; delete [] wr; delete [] wi;
        delete [] vr; delete [] evo; delete [] work;
}
```

只算右 eigenvectors

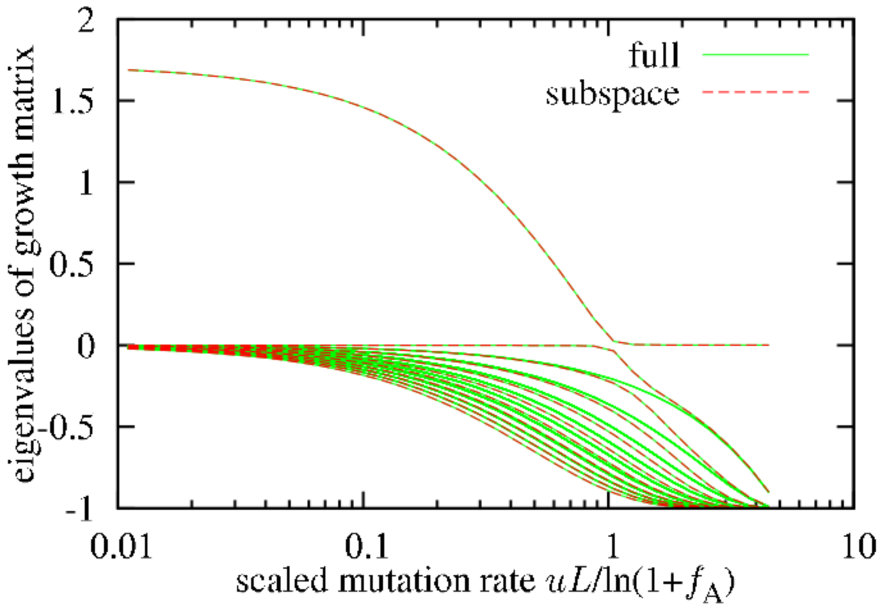
Projection to symmetric subspace

As the dynamics of the system defined above is invariant under permutation of the base pairs in a sequence, it can be *properly projected* to the symmetric subspace where sequences in the population are only characterized by their hamming distance h from the dominant sequence.

Consider the transition from $h = n$ to $h = m > n$. We need to mutate $m - n$ more base pairs that are identical to what are in the dominant species (*i.e.*, n base pairs) than those that differ from them (*i.e.*, $L - n$ base pairs) in the original sequence. Summing up all different ways of mutating the base pairs, we arrive at the element of the mutation matrix

$$\begin{aligned}
\frac{q_{m,n}}{(1-u)^L} &= C^{L-n}_{m-n} C^n_0 \left(\frac{u}{1-u}\right)^{m-n} + C^{L-n}_{m+1-n} C^n_1 \left(\frac{u}{1-u}\right)^{m-n+2} \\
&\quad + \cdots + C^{L-n}_{L-n} C^n_{L-m} \left(\frac{u}{1-u}\right)^{2L-m-n} \\
&= \sum_{i=0}^{L-m} C^{L-n}_{m-n+i} C^n_i \left(\frac{u}{1-u}\right)^{m-n+2i} \\
&= \sum_{i=0}^{L-m} \frac{(L-n)!n!}{(m-n+i)!\,(L-m-i)!i!\,(n-i)!} \left(\frac{u}{1-u}\right)^{m-n+2i} \qquad (1)
\end{aligned}$$

while the growth matrix is now $F_{m,n} = (f_A \delta_{m,0} + 1)\,\delta_{m,n}$. We can easily calculate the eigen system of such a matrix up to $L = 170$. Following is a plot of the eigen values of the reduced $(L+1) \times (L+1)$ matrix compared with the original full $2^L \times 2^L$ matrix at $L = 10$.



We see an exact match of the eigen values of the reduced matrix to those of the full matrix and some additional eigenvalues of the full matrix not matched by the reduced ones that are likely corresponding to non-symmetric modes of decay.

本週習題 (optional)

- 二維 Ising 模型問題的更正： critical point 是在 $\beta \approx 0.44$ 附近。範圍改正後，可用較小系統計算。
- 用前列的 mutation 程式把最大的 10 個 eigenvalues 對 mutation rate 作圖。
- 用 LAPACK 來計算二維的 potential well 問題。 (大概可算到 32×32 個格點，可當 final project)
- 查閱網上 ARPACK 文件，用它來計算二維的 potential well 問題。 (可算到 128×128 個格點，可當 final project)
- 證明隨機漫步在 $p = 1 / 2$ 時平均到家時間是： $\tau(x) = 2\,L\,x / 3 - x^2 / 3$ 。