Lecture 11: Linear Algebra

Sets of simultaneous equations

Eigenvalue problem

Revisiting the Schrodinger equation without the shooting method

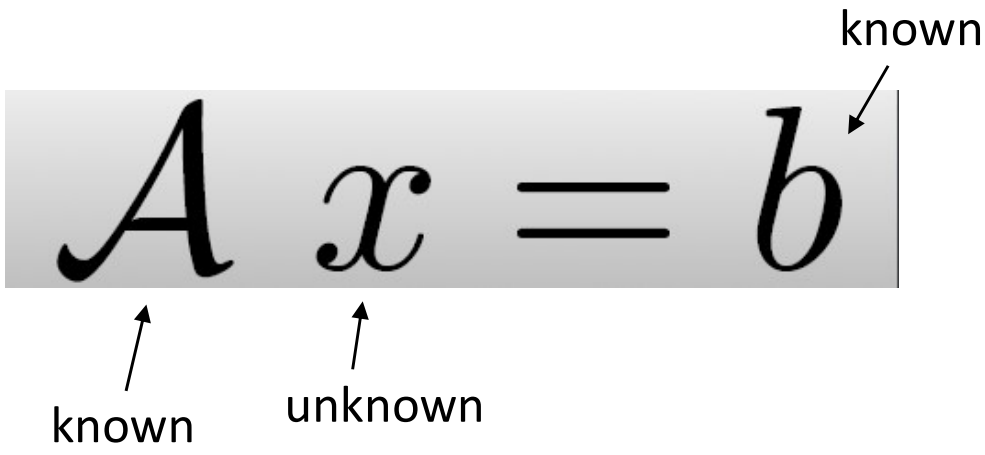
Simultaneous Sets of Linear Equations

Linear algebra (**scipy.linalg**)

Linear algebra functions.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\
a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4
\end{aligned}
$$

N linear equations, N unknowns (solvable for linearly independent equations)



$$\mathcal{A}\,x = b$$

Many solvers depending on properties of matrix **A**.

solve(a, b[, sym_pos, lower, overwrite_a, …])	Solves the linear equation set a * x = b for the unknown x for square a matrix.
solve_banded(l_and_u, ab, b[, overwrite_ab, …])	Solve the equation a x = b for x, assuming a is banded matrix.
solveh_banded(ab, b[, overwrite_ab, …])	Solve equation a x = b.
solve_circulant(c, b[, singular, tol, …])	Solve C x = b for x, where C is a circulant matrix.
solve_triangular(a, b[, trans, lower, …])	Solve the equation *a x = b* for *x*, assuming a is a triangular matrix.

$$
\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} =
\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}
$$

You want to pick the solver which is the ***least*** general for your matrix.

If Matrix has symmetry, can be taken advantage of to speed up solution.

Eigenvalue Problem

unknown unknown

$$Ax = \lambda x$$

known unknown

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

- Input
  - matrix
- Output
  - Eigenvalues
  - Eigenvectors

Linear algebra (`scipy.linalg`)

Linear algebra functions.

eig(a[, b, left, right, overwrite_a, …])	Solve an ordinary or generalized eigenvalue problem of a square matrix.
eigvals(a[, b, overwrite_a, check_finite, …])	Compute eigenvalues from an ordinary or generalized eigenvalue problem.
eigh(a[, b, lower, eigvals_only, …])	Solve a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eigvalsh(a[, b, lower, overwrite_a, …])	Solves a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eig_banded(a_band[, lower, eigvals_only, …])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigvals_banded(a_band[, lower, …])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigh_tridiagonal(d, e[, eigvals_only, …])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.
eigvalsh_tridiagonal(d, e[, select, …])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.

Tridiagonal matrix

tridag.h

```
void tridag(VecDoub_I &a, VecDoub_I &b, VecDoub_I &c, VecDoub_I &r, VecDoub_O &u)
```

Solves for a vector `u[0..n-1]` the tridiagonal linear set given by equation (2.4.1). `a[0..n-1]`, `b[0..n-1]`, `c[0..n-1]`, and `r[0..n-1]` are input vectors and are not modified.

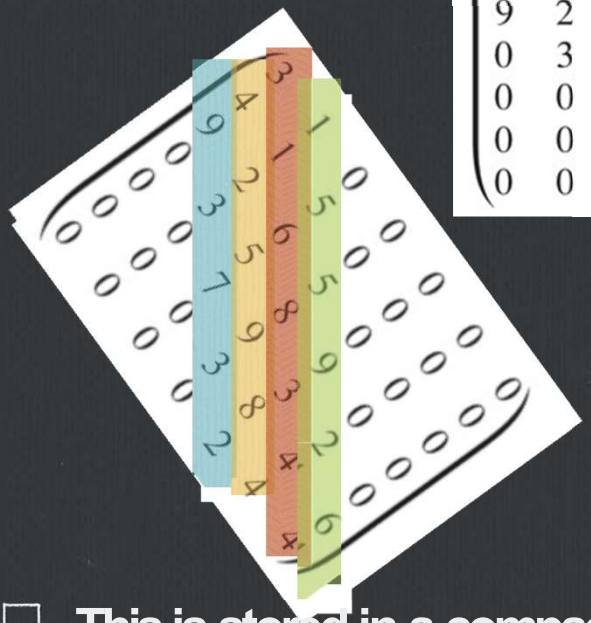
$$\begin{bmatrix} b_0 & c_0 & 0 & \cdots & & & \\ a_1 & b_1 & c_1 & \cdots & & & \\ & & & \cdots & & & \\ & & & \cdots & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & \cdots & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \cdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \cdots \\ r_{N-2} \\ r_{N-1} \end{bmatrix}$$

Need only store vectors a,b,c, and r.

- The tridiagonal matrix algorithm scales as $O(N)$, instead of $O(N^3)$.

Band-diagonal

$$\begin{pmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 5 & 0 & 0 & 0 & 0 \\ 9 & 2 & 6 & 5 & 0 & 0 & 0 \\ 0 & 3 & 5 & 8 & 9 & 0 & 0 \\ 0 & 0 & 7 & 9 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 8 & 4 & 6 \\ 0 & 0 & 0 & 0 & 2 & 4 & 4 \end{pmatrix}$$



$$\begin{pmatrix} x & x & 3 & 1 \\ x & 4 & 1 & 5 \\ 9 & 2 & 6 & 5 \\ 3 & 5 & 8 & 9 \\ 7 & 9 & 3 & 2 \\ 3 & 8 & 4 & 6 \\ 2 & 4 & 4 & x \end{pmatrix}$$

- This is stored in a compact form

Example from SciPy

scipy.linalg.solve_banded

scipy.linalg.**solve_banded**(*l_and_u*, *ab*, *b*, *overwrite_ab=False*, *overwrite_b=False*, *debug=None*, *check_finite=True*) [source]

Solve the equation a x = b for x, assuming a is banded matrix.

The matrix a is stored in *ab* using the matrix diagonal ordered form:

```
ab[u + i - j, j] == a[i,j]
```

Examples

Solve the banded system a x = b, where:

```
    [5  2 -1  0  0]       [0]
    [1  4  2 -1  0]       [1]
a = [0  1  3  2 -1]   b = [2]
    [0  0  1  2  2]       [2]
    [0  0  0  1  1]       [3]
```

There is one nonzero diagonal below the main diagonal (l = 1), and two above (u = 2). The diagonal banded form of the matrix is:

```
     [*  * -1 -1 -1]
ab = [*  2  2  2  2]
     [5  4  3  2  1]
     [1  1  1  1  *]
```

```
>>> from scipy.linalg import solve_banded
>>> ab = np.array([[0,  0, -1, -1, -1],
...                [0,  2,  2,  2,  2],
...                [5,  4,  3,  2,  1],
...                [1,  1,  1,  1,  0]])
>>> b = np.array([0, 1, 2, 2, 3])
>>> x = solve_banded((1, 2), ab, b)
>>> x
array([-2.37288136,  3.93220339, -4.        ,  4.3559322 , -1.3559322 ])
```

Linear Algebra and HPC

TOP 500
The List.

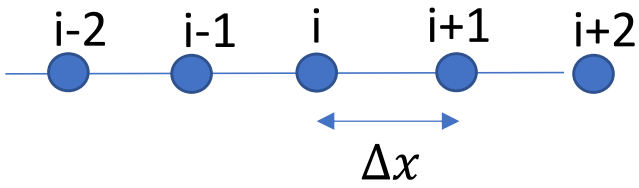
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	706,304	64,590.0	89,794.5	2,528

Lapack (linear algebra package)
BLAS (basic linear algebra subprograms)

Highly optimized linear algebra Routines for C/Fortran

From mathematics to computer: finite differences

real space grid: $f(x) \rightarrow f_i$



Forward Formula:

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+h) - f(x)}{h} + o(h) \rightarrow \frac{\partial f(x)}{\partial x} = \frac{f_{i+1} - f_i}{\Delta x}$$

Centered Formulas:

$$\frac{\partial f(x)}{\partial x} = \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h} + o(h^2) \rightarrow \frac{\partial f(x)}{\partial x} = \frac{\partial f_i}{\partial x} = \frac{f_{i+1} - f_{i-1}}{2\Delta x}$$

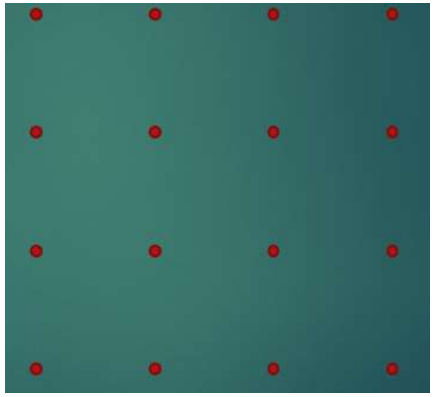
$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + o(h^2) \rightarrow \frac{\partial^2 f(x)}{\partial x^2} = \frac{f_{i+1} + f_{i-1} - 2f_i}{\Delta x^2}$$

Application to PDEs

$$\frac{\partial f(x)}{\partial x} = \frac{f_{i+1} - f_i}{\Delta x} \quad \text{(FD)}$$

$$\frac{\partial f(x)}{\partial x} = \frac{f_{i+1} - f_{i-1}}{2\Delta x} \quad \text{(CD)}$$

$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{f_{i+1} + f_{i-1} - 2f_i}{\Delta x^2}$$



Wave Equation

$$\nabla^2 U(x,t) = \frac{1}{c^2}\frac{\partial^2 U}{\partial t^2}$$

$$\frac{\partial^2 U(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 U}{\partial t^2}$$

$$\frac{U_{i+1,j} + U_{i-1,j} - 2U_{i,j}}{\Delta x^2} = \frac{1}{c^2}\frac{U_{i,j+1} + U_{i,j-1} - 2U_{i,j}}{\Delta t^2}$$

Poisson Equation: $U(x,y) \rightarrow U_{i,j}$

$$\nabla^2 U(x,y) = -4\pi\rho(x,y)$$

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = -4\pi\rho(x,y)$$

$$\frac{U_{i+1,j} + U_{i-1,j} - 2U_{i,j}}{\Delta x^2} + \frac{U_{i,j+1} + U_{i,j-1} - 2U_{i,j}}{\Delta y^2} = -4\pi\rho_{i,j}$$

Heat Equation

$$\nabla^2 U(x,t) = \frac{a\partial U}{\partial t}$$

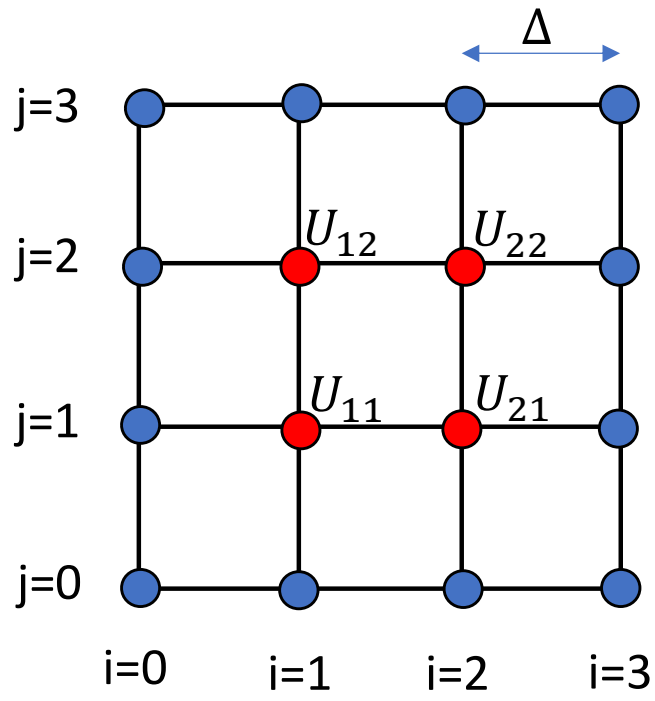
$$\frac{\partial^2 U(x,t)}{\partial x^2} = \frac{a\partial U}{\partial t}$$

$$\frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{\Delta x^2} = a\frac{f_{i,j+1} - f_{i,j}}{\Delta t}$$

Poisson Equation:

$$\frac{U_{i+1,j} + U_{i-1,j} - 2U_{i,j}}{\Delta x^2} + \frac{U_{i,j+1} + U_{i,j-1} - 2U_{i,j}}{\Delta y^2} = -4\pi\rho_{i,j} \qquad (\Delta x = \Delta y = \Delta)$$

$$\frac{1}{4}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}) - U_{i,j} = -4\pi\Delta^2\rho_{i,j} \qquad (\rho = 0 \rightarrow Laplace\ equation)$$



$$U_{11} = 1/4(U_{10} + U_{12} + U_{01} + U_{21})$$
$$U_{21} = 1/4(U_{20} + U_{22} + U_{31} + U_{11})$$
$$U_{12} = 1/4(U_{11} + U_{13} + U_{02} + U_{22})$$
$$U_{22} = 1/4(U_{21} + U_{23} + U_{12} + U_{32})$$

Use matrix solver to simultaneously solve for the potential at each grid point.

boundary conditions: blue points

unknown potential: red points

$$A \qquad x \quad = \quad b$$

$$\begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \begin{pmatrix} U_{11} \\ U_{21} \\ U_{12} \\ U_{22} \end{pmatrix} = \begin{pmatrix} U_{10} + U_{01} \\ U_{20} + U_{31} \\ U_{13} + U_{02} \\ U_{23} + U_{32} \end{pmatrix}$$

Schrodinger equation (1D)

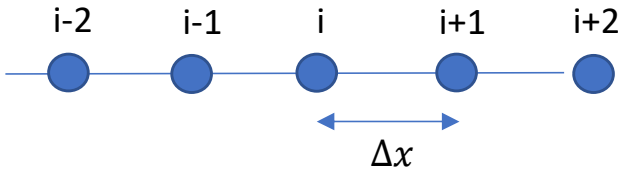
$$-\frac{1}{2}\frac{\partial^2\Psi(x)}{\partial x^2}+V(x)\Psi(\mathrm{x})=\mathrm{E}\Psi(x)$$

$$f(x)\rightarrow f_i$$

$$\frac{\partial^2 f(x)}{\partial x^2}=\frac{\mathrm{f_{i+1}+f_{i-1}-2f_i}}{\Delta x^2}$$

$$-\frac{1}{2}\frac{(\Psi_{i+1}+\Psi_{i-1}-2\Psi_i)}{\Delta x^2}+V_i\Psi_i=E\Psi_i$$

(equation for each grid point → N equations)



$$i=0:\quad -\frac{1}{2}\frac{(\Psi_1+\Psi_{-1}-2\Psi_0)}{\Delta x^2}+V_0\Psi_0=E\Psi_0$$

$$i=1:\quad -\frac{1}{2}\frac{(\Psi_2+\Psi_0-2\Psi_1)}{\Delta x^2}+V_1\Psi_1=E\Psi_1$$

$$i=2:\quad -\frac{1}{2}\frac{(\Psi_3+\Psi_1-2\Psi_2)}{\Delta x^2}+V_2\Psi_2=E\Psi_2$$

$$i=3:\quad -\frac{1}{2}\frac{(\Psi_4+\Psi_2-2\Psi_3)}{\Delta x^2}+V_3\Psi_3=E\Psi_3$$

…

What is the matrix form of this set of equations??

Kinetic Energy

Potential Energy

$$-\frac{1}{2\Delta x^2}\begin{pmatrix}-2 & 1 & 0 & 0 & 0\\ 1 & -2 & 1 & 0 & 0\\ 0 & 1 & -2 & 1 & 0\\ 0 & 0 & 1 & -2 & 1\\ 0 & 0 & 0 & 1 & -2\end{pmatrix}\begin{pmatrix}\Psi_0\\ \Psi_1\\ \Psi_2\\ \Psi_3\\ \Psi_4\end{pmatrix}+\begin{pmatrix}V_0 & 0 & 0 & 0 & 0\\ 0 & V_1 & 0 & 0 & 0\\ 0 & 0 & V_2 & 0 & 0\\ 0 & 0 & 0 & V_3 & 0\\ 0 & 0 & 0 & 0 & V_4\end{pmatrix}\begin{pmatrix}\Psi_0\\ \Psi_1\\ \Psi_2\\ \Psi_3\\ \Psi_4\end{pmatrix}=E\begin{pmatrix}\Psi_0\\ \Psi_1\\ \Psi_2\\ \Psi_3\\ \Psi_4\end{pmatrix}$$

$$i=0:\quad -\frac{1}{2}\frac{(\Psi_1+\Psi_{-1}-2\Psi_0)}{\Delta x^2}+V_0\Psi_0=E\Psi_0$$

$$i=1:\quad -\frac{1}{2}\frac{(\Psi_2+\Psi_0-2\Psi_1)}{\Delta x^2}+V_1\Psi_1=E\Psi_1$$

$$i=2:\quad -\frac{1}{2}\frac{(\Psi_3+\Psi_1-2\Psi_2)}{\Delta x^2}+V_2\Psi_2=E\Psi_2$$

$$i=3:\quad -\frac{1}{2}\frac{(\Psi_4+\Psi_2-2\Psi_3)}{\Delta x^2}+V_3\Psi_3=E\Psi_3$$

…

Eigenvalue problem:

$$\hat{H}\Psi=E\Psi$$

Hamiltonian matrix is tridiagonal and Hermitian.

In 1-D, with a real space basis, the Laplacian ∇^2 can be represented as:

$$\frac{1}{\Delta x^2}\begin{pmatrix}-2 & 1 & 0 & 0 & 0\\ 1 & -2 & 1 & 0 & 0\\ 0 & 1 & -2 & 1 & 0\\ 0 & 0 & 1 & -2 & 1\\ 0 & 0 & 0 & 1 & -2\end{pmatrix}$$

Setting up matrix

$$-\frac{1}{2\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix}$$

for N=5

```
np.diag(np.ones(N-1),k=-1)
```

```
array([[0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

```
np.diag(np.ones(N-1),k=0)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
np.diag(np.ones(N-1),k=1)
```

```
array([[0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0.]])
```

```
np.diag(np.ones(N-1),k=-1)+np.diag(-2*np.ones(N),k=0)+np.diag(np.ones(N-1),k=1)
```

```
array([[-2.,  1.,  0.,  0.,  0.],
       [ 1., -2.,  1.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  1., -2.,  1.],
       [ 0.,  0.,  0.,  1., -2.]])
```

```
L=1  #actual length of x domain
N=10 #number of x grid points
dx= L/(N-1)  # space between grid points

KE=(np.diag(-2*np.ones(N))+np.diag(np.ones(N-1),k=1)+np.diag(np.ones(N-1),k=-1))
KE=-0.5*KE/(dx**2)
print(KE)
```

```
[[ 81.  -40.5  -0.   -0.   -0.   -0.   -0.   -0.   -0.   -0. ]
 [-40.5  81.  -40.5  -0.   -0.   -0.   -0.   -0.   -0.   -0. ]
 [ -0.  -40.5  81.  -40.5  -0.   -0.   -0.   -0.   -0.   -0. ]
 [ -0.   -0.  -40.5  81.  -40.5  -0.   -0.   -0.   -0.   -0. ]
 [ -0.   -0.   -0.  -40.5  81.  -40.5  -0.   -0.   -0.   -0. ]
 [ -0.   -0.   -0.   -0.  -40.5  81.  -40.5  -0.   -0.   -0. ]
 [ -0.   -0.   -0.   -0.   -0.  -40.5  81.  -40.5  -0.   -0. ]
 [ -0.   -0.   -0.   -0.   -0.   -0.  -40.5  81.  -40.5  -0. ]
 [ -0.   -0.   -0.   -0.   -0.   -0.   -0.  -40.5  81.  -40.5]
 [ -0.   -0.   -0.   -0.   -0.   -0.   -0.   -0.  -40.5  81. ]]
```

No potential → infinite square well

Solving

(N=10)

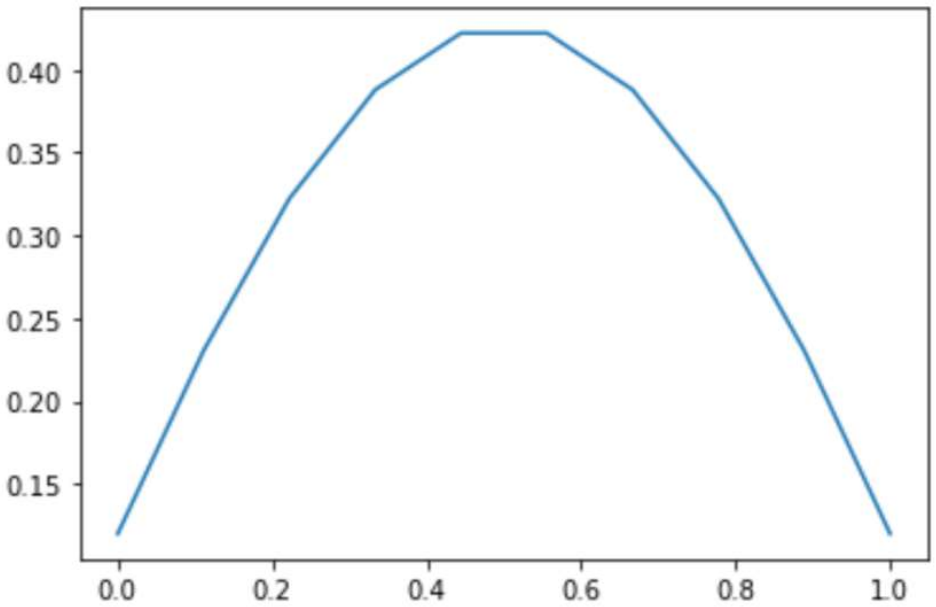
```
from scipy import linalg

xs=delta*np.arange(N)
w,v=linalg.eigh(KEmatrix) # w=eigenvalues, v = eigenvectors
```

```
w
```

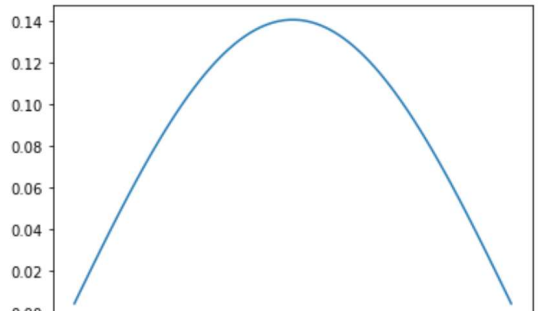
```
array([  3.28106914,  12.85846384,  27.95628055,  47.35138395,
        69.4724981 ,  92.5275019 , 114.64861605, 134.04371945,
       149.14153616, 158.71893086])
```

```
plt.plot(xs,v[:,0])
plt.show()
```



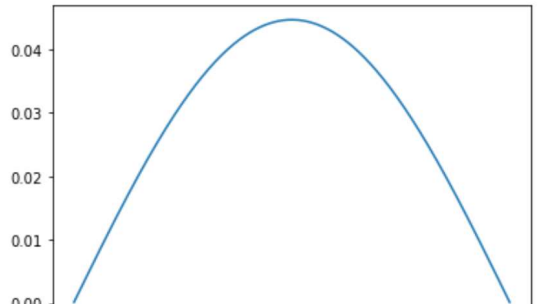

```
print(N)
print(w[0])
plt.plot(xs,v[:,0])
plt.show()
```

```
100
4.7409172562290305
```



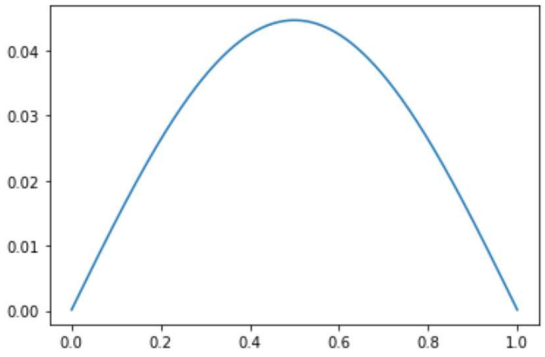

```
print(N)
print(w[0])
plt.plot(xs,v[:,0])
plt.show()
```

```
1000
4.915098376750613
```



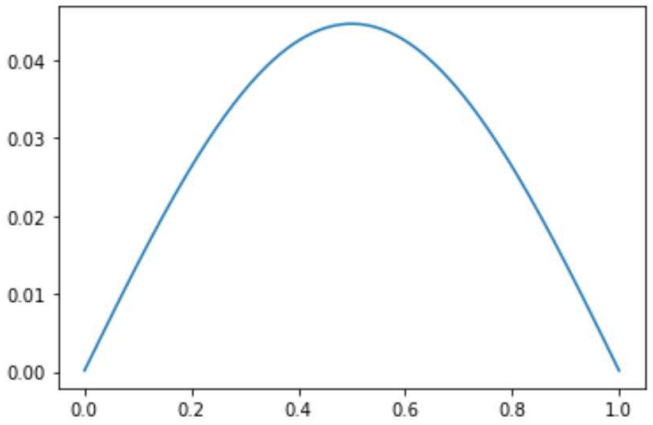

```
print(N)
print(w[0])
plt.plot(xs,v[:,0])
plt.show()
```

```
10000
4.915098376750613
```



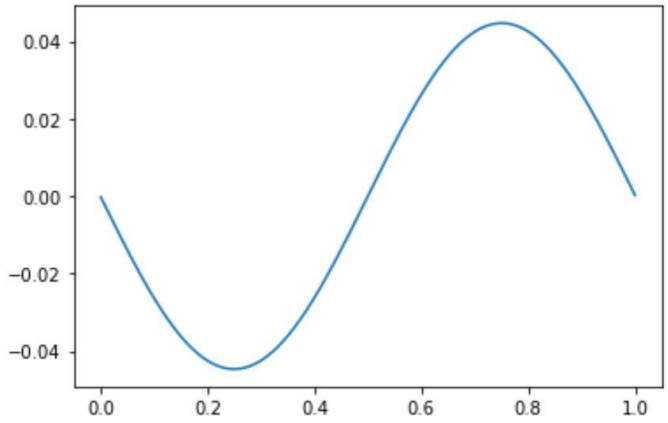
```
print(N)
print(w[0])
plt.plot(xs,v[:,0])
plt.show()
```

```
1000
4.915098376750613
```




```
print(N)
print(w[1])
plt.plot(xs,v[:,1])
plt.show()
```

```
1000
19.66034509328222
```




```
print(N)
print(w[2])
plt.plot(xs,v[:,2])
plt.show()
```

```
1000
44.235594911021856
```