UNIVERSITY OF TEXAS ARLINGTON

PHYS 5319-001: Math Methods in Physics III

Eigenvalue Problems and Optimization

Instructor:	Dr. Qiming Zhang
Office:	CPB 336
Phone:	817-272-2020
Email:	zhang@uta.edu

Stationary Schrödinger equation

$$\hat{H}\varphi(\boldsymbol{r}) = E\varphi(\boldsymbol{r})$$

where $\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(x)$.

Bound state $E \rightarrow E_n, \{\varphi_n\}$

In general, a Hermitian matrix H: $\quad H\boldsymbol{v} = E\boldsymbol{v}$

Or $\quad (H - EI)v = 0 \quad$ where I is a unit matrix

To have , we must have $\det(H - EI) = 0$

$$\begin{bmatrix} H_{11}-E & H_{12} & H_{13} & H_{14} & \cdots \\ H_{21} & H_{22}-E & H_{23} & H_{24} & \cdots \\ H_{31} & H_{32} & H_{33}-E & H_{34} & \cdots \\ H_{41} & H_{42} & H_{43} & H_{44}-E & \cdots \\ & & \vdots & & \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \psi_4 \\ \vdots \end{bmatrix} = 0$$

To solve

$$\hat{H}|\varphi>= E|\varphi>, \quad \varphi(\boldsymbol{r}) \equiv |\varphi>$$

For a complete basis $\{|\phi_n>\}$, simply let $|\phi_n> \equiv |n>$.
Orthonormal: $<n|m>= \delta_{nm}$
Complete: $\sum_n |n><n| = I$

Expand any state $|\varphi>= \sum_n a_n|n>$, and substitute it into Schrödinger equation:

$$\int \phi_m H \phi_n \, d\tau = H_{mn}$$

$$\hat{H}\sum_n \underline{a_n}|n> = E\sum_n a_n|n> ,$$

a_n

"scalar product" $<m|$ from left: $\sum_n <\underline{m|\hat{H}|n}>= E a_m$

Since m could be 1, 2, …, we have an eigen equation

$$\begin{pmatrix} H \end{pmatrix}\begin{pmatrix} a_1 \\ \vdots \end{pmatrix} = E\begin{pmatrix} a \end{pmatrix}$$

$$H\boldsymbol{a} = E\boldsymbol{a}$$

where $\boldsymbol{a}$ is a vector with $\{a_n\}$ elements

eigen-problem $H\boldsymbol{a} = E\boldsymbol{a}$

We need to solve $\det(H - EI) = 0$

for $E_1, E_2, \ldots, E_i, \ldots$ (real value)
and (optionally) eigenvectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_i, \ldots$

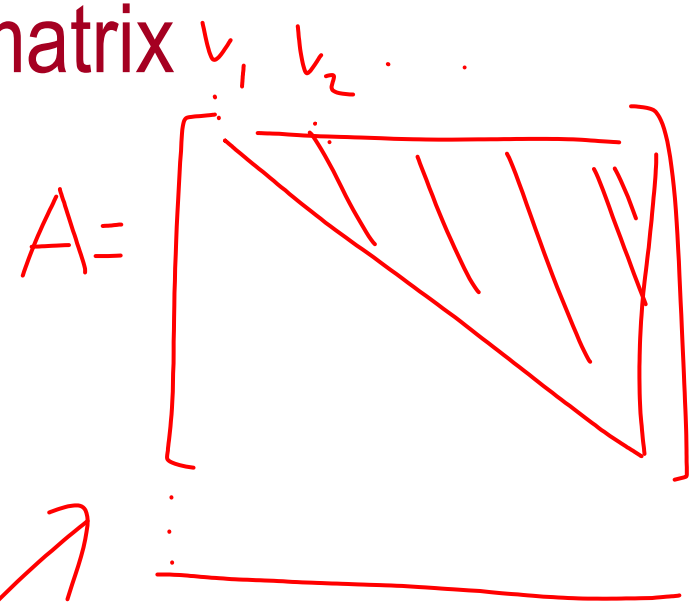
Matrix H could be real symmetric: $H_{nm} = H_{mn}$
or complex Hermitian: $H_{nm} = H_{mn}^*$

EISPACK is a software library for numerical computation of eigenvalues and eigenvectors of matrices, written in FORTRAN.

It is usually included in LAPACK

real symmetric matrix

```
subroutine ssyev ( character                JOBZ,
                   character                UPLO,
                   integer                  N,
                   real, dimension( lda, * ) A,
                   integer                  LDA,
                   real, dimension( * )     W,
                   real, dimension( * )     WORK,
                   integer                  LWORK,
                   integer                  INFO
                 )
```



My FORTRAN code:

```
call ssyev('V', 'U', n, a, n, w, work, lwork, info)
print*,'info= ', info
do i=1,10
  print*,w(i)
enddo
```

Complex Hermitian matrix

cheev()

```
subroutine cheev ( character                    JOBZ,
                   character                    UPLO,
                   integer                      N,
                   complex, dimension( lda, * ) A,
                   integer                      LDA,
                   real, dimension( * )         W,
                   complex, dimension( * )      WORK,
                   integer                      LWORK,
                   real, dimension( * )         RWORK,
                   integer                      INFO
                 )
```

λ_i

CHEEV computes the eigenvalues and, optionally, the left and/or right eigenvectors for HE matrices

- in LAPACK

On MATLAB

eig

Eigenvalues and eigenvectors

Syntax

```
e = eig(A)
[V,D] = eig(A)
[V,D,W] = eig(A)
```

```
e = eig(A,B)
[V,D] = eig(A,B)
[V,D,W] = eig(A,B)
```

```
[ __ ] = eig(A,balanceOption)
[ __ ] = eig(A,B,algorithm)
```

```
[ __ ] = eig( __ ,eigvalOption)
```

Description

`e = eig(A)` returns a column vector containing the eigenvalues of square matrix `A`.

`[V,D] = eig(A)` returns diagonal matrix `D` of eigenvalues and matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = V*D`.

python

scipy.linalg.eig

The function `scipy.linalg.eig` computes eigenvalues and eigenvectors of a square matrix A.

Let's consider a simple example with a diagonal matrix:

```
A = np.array([[1,0],[0,-2]])
print(A)
```

```
[[ 1  0]
 [ 0 -2]]
```

The function `la.eig` returns a tuple `(eigvals,eigvecs)` where `eigvals` is a 1D NumPy array of complex numbers giving the eigenvalues of A, and `eigvecs` is a 2D NumPy array with the corresponding eigenvectors in the columns:

```
results = la.eig(A)
```

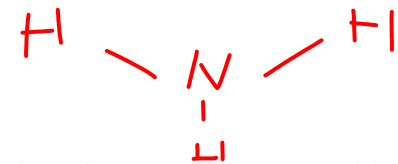
The eigenvalues of A are:

```
print(results[0])
```

```
[ 1.+0.j -2.+0.j]
```

$\sum_i \phi_i^{\dagger}$

If the basis $|\phi_n >\equiv |n >$ is not orthogonal: $< n|m >\equiv S_{nm}$ (but definitely $S_{nn} > 0$)



e.g. In Quantum Chemistry, the atomic orbitals are usually used as a basis. For NH_3 molecule calculations, you need to include: 1s, 2s, 2p,… orbitals for **each** Hydrogen atom;
2s, 2p, 3s, 3p, … for the Nitrogen atom. These orbitals are not orthogonal if from different atoms (overlap).

$$S = \begin{pmatrix} S_{11} & & \\ & \ddots & \\ & & \ddots \end{pmatrix}$$

$$< m| \quad \Rightarrow \quad \hat{H}\sum_n a_n|n> \; = E\sum_n a_n|n>$$

Obtain:

$$H\boldsymbol{a} = E\underline{S}\boldsymbol{a}$$

where S is a positive-definite matrix

called generalized eigenvalue problem

Like a positive number could be expressed as a square, we can let $S = LL^T$, where "T" refers to "transpose".

$L^{-1}L = I$

$(LL^{-1})^T$

Left multiply L^{-1} to

$$H\boldsymbol{a} = ES\boldsymbol{a}$$

$$\Rightarrow \quad L^{-1}H(L^{-1})^T L^T\boldsymbol{a} = EL^T\boldsymbol{a}$$

Write the similar transformation $C \equiv L^{-1}H(L^{-1})^T$ still symmetric/Hermitian.

and $\boldsymbol{b} = L^T\boldsymbol{a}$. The problem becomes the familiar eigen-problem:

$$C\boldsymbol{b} = E\boldsymbol{b}$$

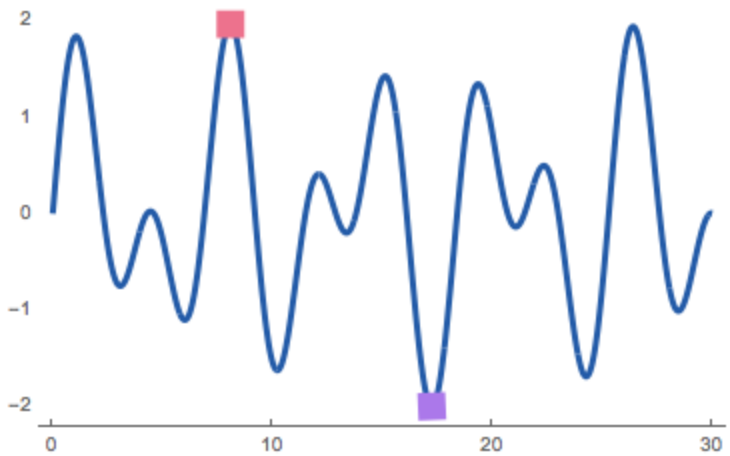
In EISPACK, you just need one call/function to do it

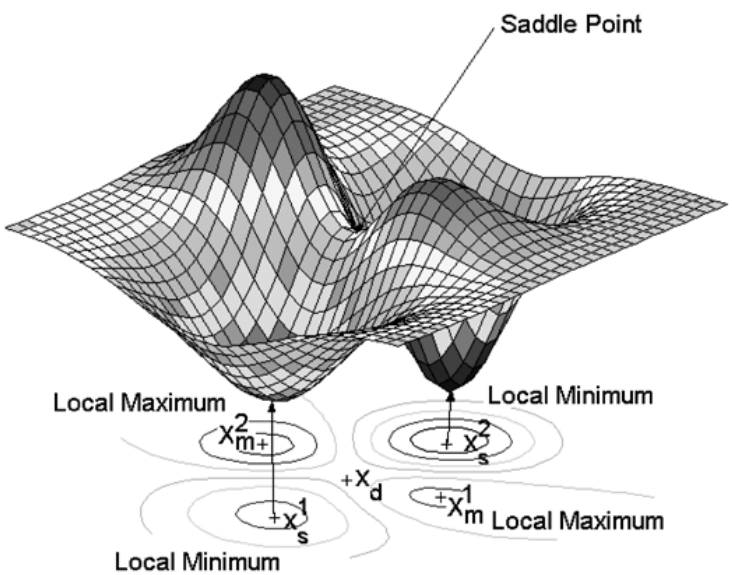
Get $\{E_i\}$ and eigenvectors $\boldsymbol{a} = (L^T)^{-1}\boldsymbol{b}$

```
subroutine chegvx ( integer                      ITYPE,
                    character                    JOBZ,
                    character                    RANGE,
                    character                    UPLO,
                    integer                      N,
                    complex, dimension( lda, * ) A,
                    integer                      LDA,
                    complex, dimension( ldb, * ) B,
                    integer                      LDB,
                    real                         VL,
                    real                         VU,
                    integer                      IL,
                    integer                      IU,
                    real                         ABSTOL,
                    integer                      M,
                    real, dimension( * )         W,
                    complex, dimension( ldz, * ) Z,
                    integer                      LDZ,
                    complex, dimension( * )      WORK,
                    integer                      LWORK,
                    real, dimension( * )         RWORK,
                    integer, dimension( * )      IWORK,
                    integer, dimension( * )      IFAIL,
                    integer                      INFO
                  )
```

Optimization

- Object function with *N* parameters $f(x_1, x_2, \dots)$
- Minima/maxima search on a *N-d* space surface
- local *vs.* global
- We focus on minima here





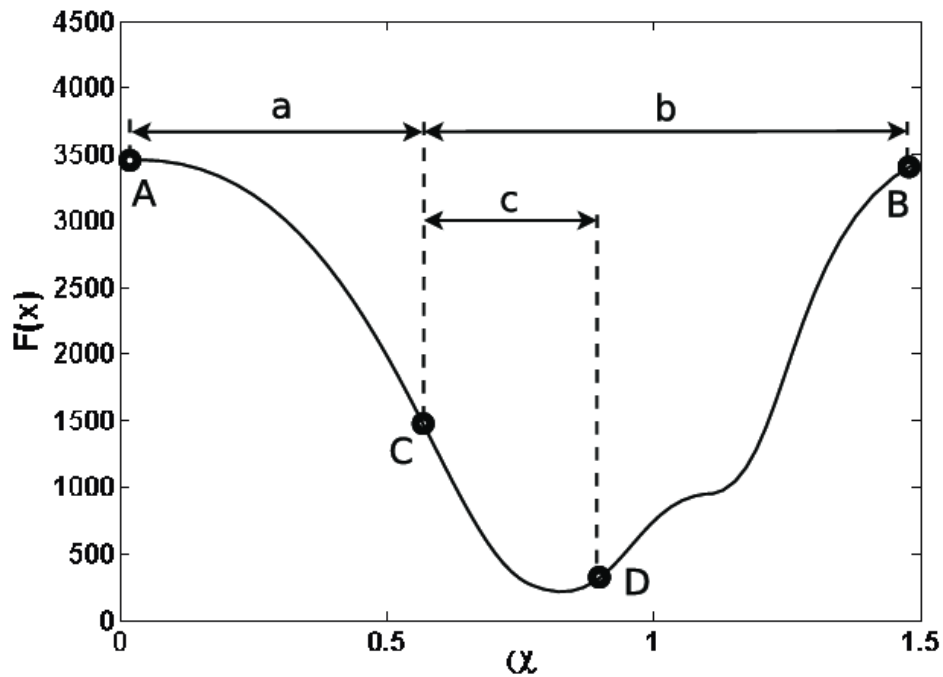
Golden section search for 1-d

$\frac{a_n}{a_{n+1}}$

- Triplet of points: A, C, & B (choose C: A<C<B)

$\frac{a}{a+b} = \frac{3-\sqrt{5}}{2} = 0.38197$, and $\frac{b}{a+b} = 0.618$

- If F(C) < F(A) & F(B), then a minimum exist.
- Find a new point x either in (A,C) or (C,B)
- If F(C)<F(x)⇒(A,C,x)
- If F(C)>F(x) ⇒(C,x,B)
- continue till tolerance reachec



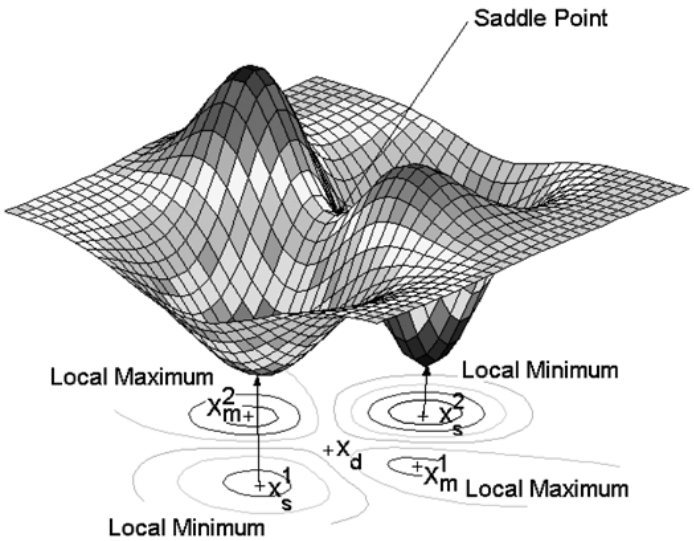
Given the triplet points, the next point (x) to be tried is 0.38197 fraction into the larger of the two intervals.

Steepest descent method

- Object function with *N* parameters $f(x_1, x_2, \ldots)$ is a surface in N-dimension space. Imaging it as a potential surface (V).

$\vec{F} = -\nabla V$

- gradient $\nabla f = \frac{\partial f}{\partial x_1}\hat{e}_1 + \frac{\partial f}{\partial x_2}\hat{e}_2 + \cdots$
- Start at a point $\boldsymbol{X}^{(0)}$, set $k=0$
- $\boldsymbol{X}^{(k+1)} = \boldsymbol{X}^{(k)} \underline{- \nabla f(\boldsymbol{X}^{(k)})}\Delta t$
- Until $|\nabla f| \leq \epsilon$



- Only reach a local minimum