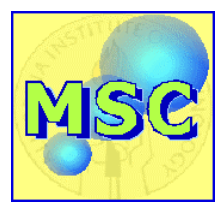
Python Short Course
Lecture 2: Numerical Python

Richard P. Muller
Materials and Process Simulation Center
May 11, 2000
121 Beckman Institute
Caltech



NumPy Modules

- NumPy has many of the features of Matlab, in a free, multiplatform program. It also allows you to do intensive computing operations in a simple way
- Numeric Module: Array Constructors
  - ones, zeros, identity
  - arrayrange
- LinearAlgebra Module: Solvers
  - Singular Value Decomposition
  - Eigenvalue, Eigenvector
  - Inverse
  - Determinant
  - Linear System Solver

© 2000 Richard P. Muller

Simple Numeric Constructors

- Arrays are slightly different from lists. They can only contain one type of data structure, and they are much faster to work with numerically.

```
>>> from Numeric import *
>>> s = arrayrange(0,2*pi,0.1) #"arange" also
>>> print s
[0., 0.1, ... 6.2]
>>> sin(s) #Numeric.sin maps onto arrays
[0., 0.099833, ... -0.0830894]
```

© 2000 Richard P. Muller

Shape and reshape

```
>>> a = zeros((3,3),Float)
>>> print a
[[0.,0.,0.],
 [0.,0.,0.],
 [0.,0.,0.]]
>>> print a.shape
(3,3)
>>> reshape(a,(9,)) # could also use a.flat
>>> print a
[0.,0.,0.,0.,0.,0.,0.,0.,0.]
```

© 2000 Richard P. Muller

Arrays and Constructors

```
>>> a = ones((3,3),Float)
>>> print a
[[1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.]]
>>> b = zeros((3,3),Float)
>>> b = b + 2.*identity(3) #"+" is overloaded
>>> c = a + b
>>> print c
[[3., 1., 1.],
 [1., 3., 1.],
 [1., 1., 3.]]
```

© 2000 Richard P. Muller

Overloaded operators

```
>>> b = 2.*ones((2,2),Float) #overloaded
>>> print b
[[2.,2.],
 [2.,2.]]
>>> b = b+1    # Addition of a scalar is
>>> print b    #   element-by-element
[[3.,3.],
 [3.,3.]]
>>> c = 2.*b   # Multiplication by a scalar is
>>> print c    #   element-by-element
[[6.,6.],
 [6.,6.]]
```

© 2000 Richard P. Muller

More on overloaded operators

```
>>> c = 6.*ones((2,2),Float)
>>> a = identity(2)
>>> print a*c
[[6.,0.],              # ARGH! element-by-element!
 [0.,6.]]
>>> matrixmultiply(a,c)
[[6.,6.],
 [6.,6.]]
```

© 2000 Richard P. Muller

Array functions

```
>>> from LinearAlgebra import *
>>> a = zeros((3,3),Float) + 2.*identity(3)
>>> print inverse(a)
[[0.5, 0., 0.],
 [0., 0.5, 0.],
 [0., 0., 0.5]]
>>> print determinant(inverse(a))
0.125
>>> print diagonal(a)
[0.5,0.5,0.5]
>>> print diagonal(a,1)
[0.,0.]
```

- transpose(a), argsort(), dot()

© 2000 Richard P. Muller

Eigenvalues

```
>>> from LinearAlgebra import *
>>> val = eigenvalues(c)
>>> val, vec = eigenvectors(c)
>>> print val
[1., 4., 1.]
>>> print vec
[[0.816, -0.408, -0.408],
 [0.575, 0.577, 0.577],
 [-0.324, -0.487, 0.811]]
```

- also solve_linear_equations, singular_value_decomposition, etc.

© 2000 Richard P. Muller

Least Squares Fitting

- Part of Hinsen's Scientific Python module

```
>>> from LeastSquares import *
>>> def func(params,x): # y=ax^2+bx+c
      return params[0]*x*x + params[1]*x +
                params[2]
>>> data = []
>>> for i in range(10):
      data.append((i,i*i))
>>> guess = (3,2,1)
>>> fit_params, fit_error =
      leastSquaresFit(func,guess,data)
>>> print fit_params
[1.00,0.000,0.00]
```

© 2000 Richard P. Muller

FFT

```
>>> from FFT import *
>>> data = array((1,0,1,0,1,0,1,0))
>>> print fft(data).real
[4., 0., 0., 0., 4., 0., 0., 0.]]
```

- Also note that the FFTW package ("fastest Fourier transform in the West") has a python wrapper. See notes at the end

© 2000 Richard P. Muller

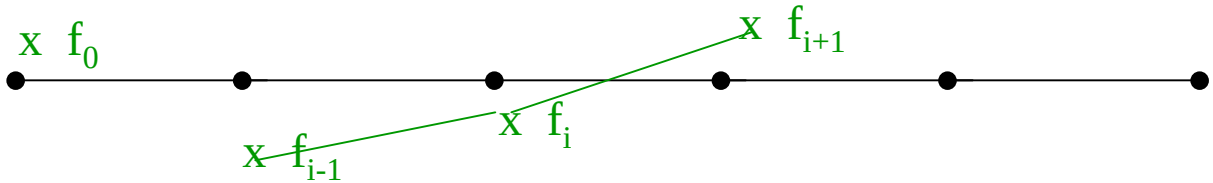
Example: Particle in a Box

```
N = 100
T = get_kinetic_energy(N)
V = get_particle_box_potential(N)
H = T + V
val, vec = eigenvectors(H)
val, vec = ev_sort(val, vec)
plot_results(vec[:2])
```

© 2000 Richard P. Muller

Finite Difference Approximation

- Consider a set of functional values on a grid



- We can calculate forward and backwards derivatives by simple differences
  - $df^- = (f_i - f_{i-1})/h$
  - $df^+ = (f_{i+1} - f_i)/h$
- We can take differences of these to get an approximation to the second derivative
  - $ddf = (df^+ - df^-)/h$
  - $ddf = (f_{i-1} - 2f_i + f_{i+1})/h^2$

© 2000 Richard P. Muller

get_kinetic_energy function

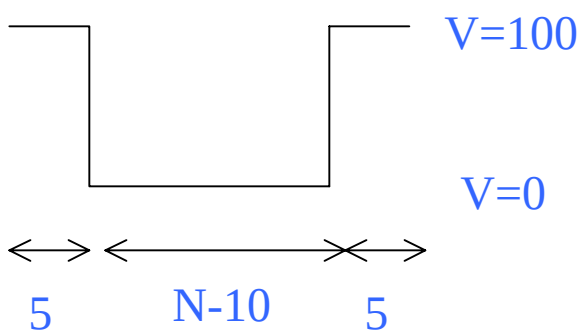
```
def get_kinetic_energy(N):
  T = zeros((N,N),Float) + identity(N)
  for i in range(N-1):
       T[i,i+1] = T[i+1,i] = -0.5
  return T
```

1.0	-0.5		
-0.5	1.0	-0.5	
	-0.5	1.0	-0.5
		-0.5	1.0

© 2000 Richard P. Muller

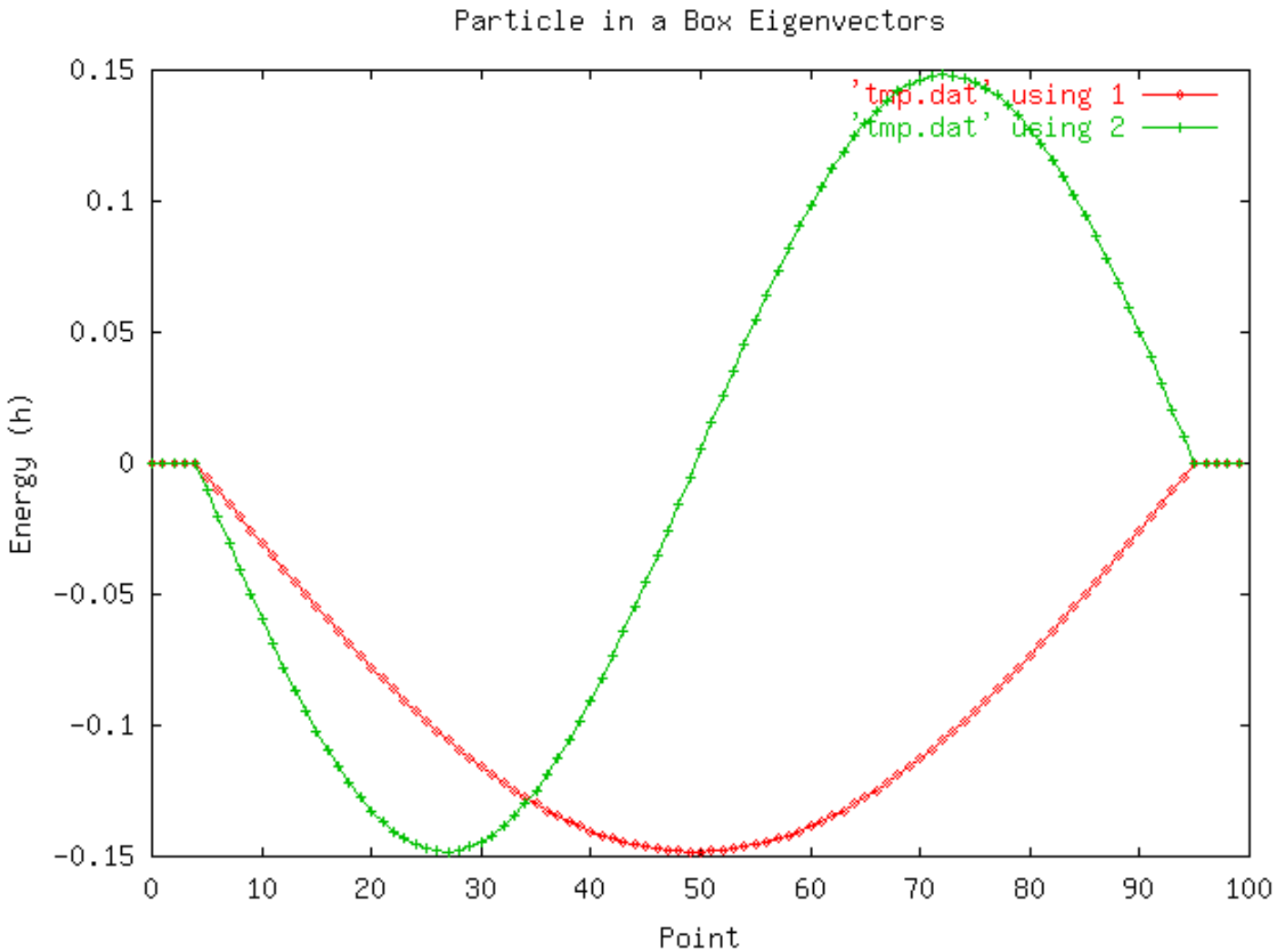
get_particle_box_potential function

```
def get_particle_box_potential(N):
  border = 5
  V = zeros((N,N),Float)
  for i in range(border):
       V[i,i] = V[N-1-i,N-1-i] = 100.
  return V
```



© 2000 Richard P. Muller

Particle in a Box Wave Function



© 2000 Richard P. Muller

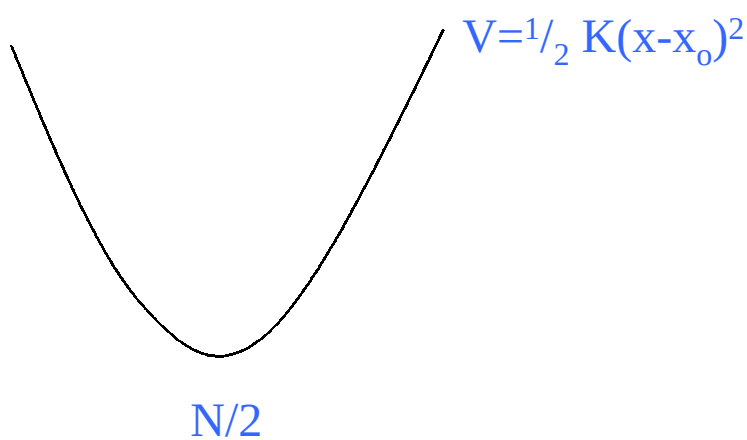
Example: Harmonic Oscillator

```
N = 100
T = get_kinetic_energy(N)
V = get_harmonic_oscillator_potential(N)
H = T + V
val, vec = eigenvectors(H)
val, vec = ev_sort(val, vec)
plot_results(vec[:2])
```

© 2000 Richard P. Muller

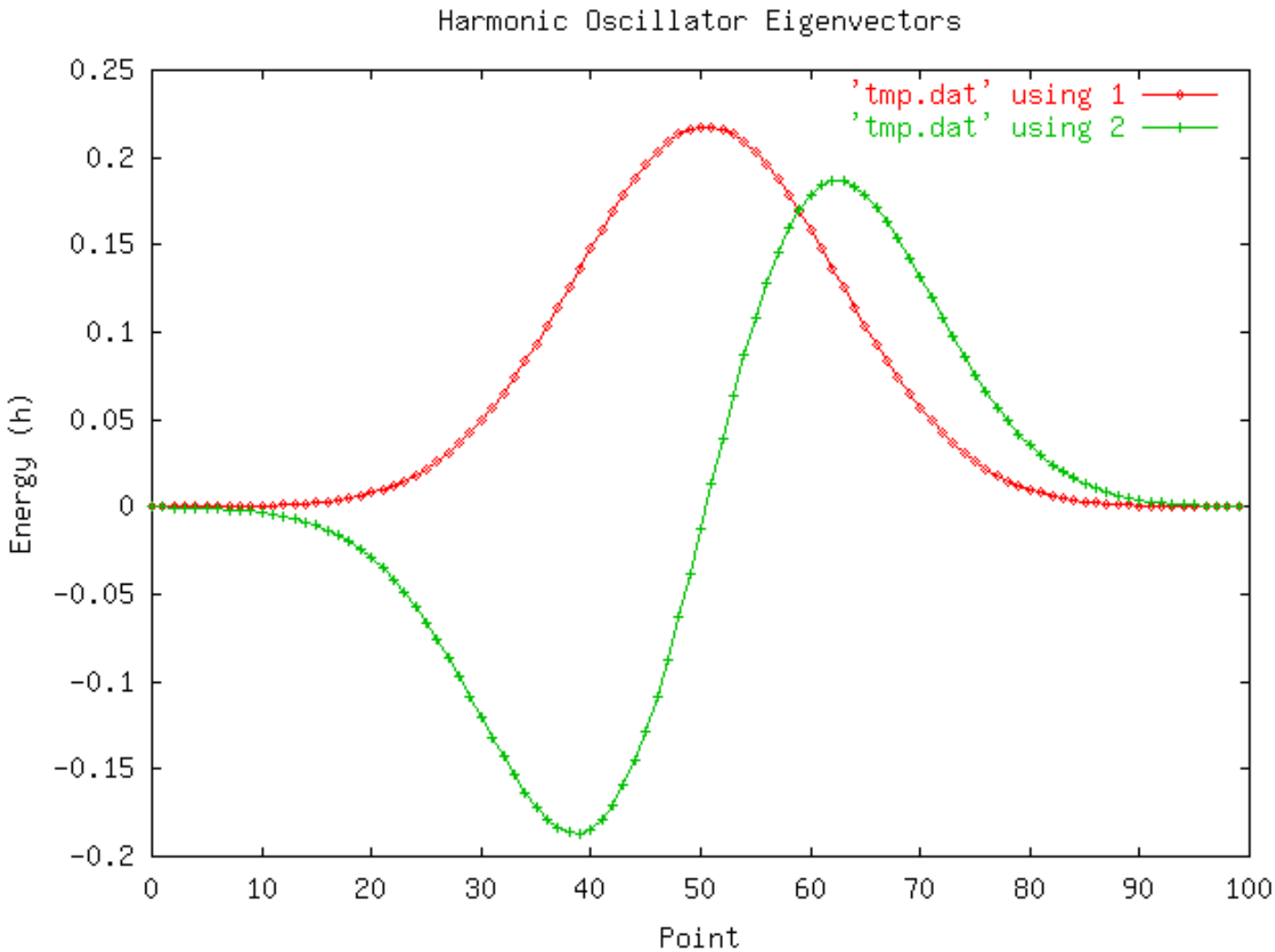
get_harmonic_oscillator_potential

```
def get_harmonic_oscillator_potential(N):
  midpoint = N/2 + 0.5
  K = 0.5/(N*N) # independent of N
  V = zeros((N,N),Float)
  for i in range(N):
       delx = i - midpoint
       V[i,i] = 0.5*K*delx*delx
  return V
```



© 2000 Richard P. Muller

Harmonic Oscillator Wave Function



© 2000 Richard P. Muller

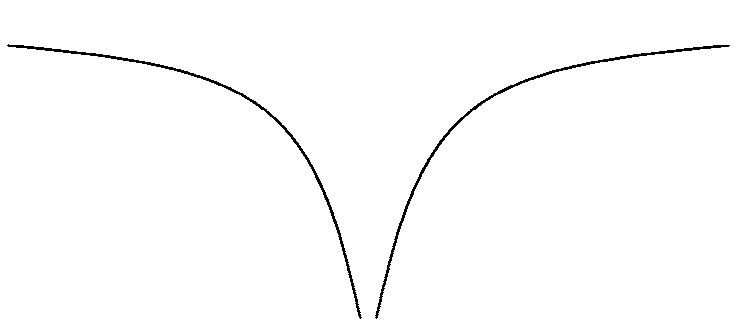
Example: One-D Hydrogen

```
N = 100
T = get_kinetic_energy(N)
V = get_oned_hydrogen_potential(N)
H = T + V
val, vec = eigenvectors(H)
val, vec = ev_sort(val, vec)
plot_results(vec[:2])
```

© 2000 Richard P. Muller

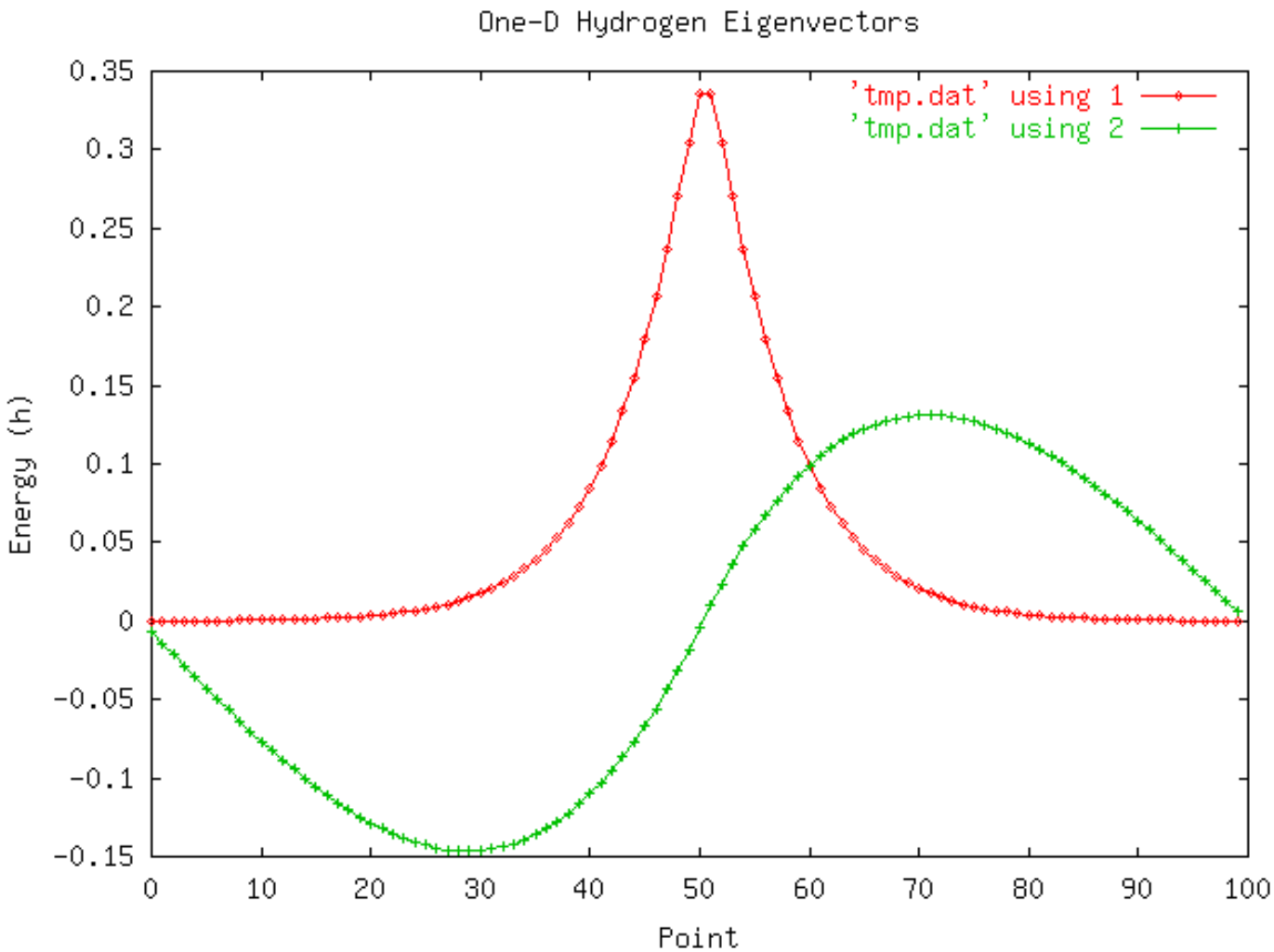
get_oned_hydrogen_potential

```
def get_oned_hydrogen_potential(N):
  midpoint = N/2 + 0.5
  Qeff = 3./N # independent of N
  V = zeros((N,N),Float)
  for i in range(N):
       delx = i - midpoint
       V[i,i] = -Qeff/abs(delx)
  return V
```



© 2000 Richard P. Muller

One-D Hydrogen Eigenvectors



© 2000 Richard P. Muller

Command Line Arguments

- We would like to be able to choose the different potentials via command-line flags, i.e.,

```
% one_d_hamiltonian.py -b -n 50
(Box wave function with 50 points)

% one_d_hamiltonian.py -s
(Spring wave function with 100 points)

% one_d_hamiltonian.py -h -n 200
(One-D H wave function with 200 points)
```

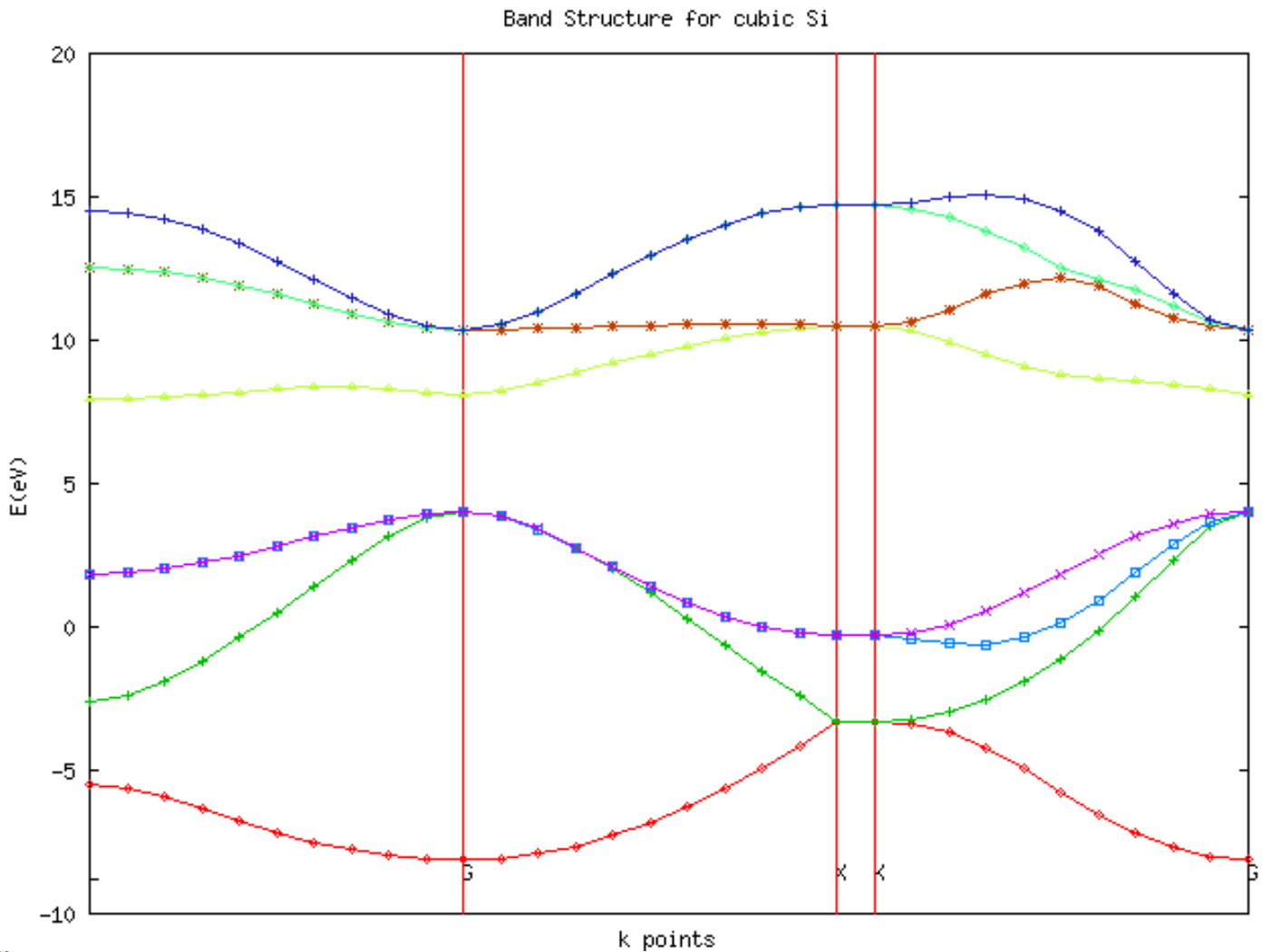
© 2000 Richard P. Muller

getopt module

```
opts, args = getopt.getopt(sys.argv[1:],'bshn:')
for opt in opts:
  key,value = opt
  if key == '-b':
       potential == 'box'
  elif key == '-s':
       potential == 'spring'
  elif key == '-h':
       potential == 'hydrogen'
  elif key == '-n':
       N = eval(value)
```

© 2000 Richard P. Muller

Example: Tight Binding Band Structure of Semiconductors



© 2000 Richard P. Muller

Tight Binding Theory

- Extended-Huckel treatment of electronic structure
  - Diagonal elements of H have a self term.
  - Off-diagonal elements of H have a term related to the coupling
- Include periodic boundary conditions
- k-points sample different space group symmetries
- Look at a sample Hamiltonian
  - Diamond, cubic Zincblend structures
  - 2 atoms per unit cell (cation and anion)
  - Minimal basis (s,p_x,p_y,p_z) functions on each atom
  - Off-diagonal coupling is modulated by phase factors

© 2000 Richard P. Muller

Harrison Hamiltonian

	s^c	s^a	p_x^c	p_y^c	p_z^c	p_x^a	p_y^a	p_z^a
s^c	E_s^c	$E_{ss}g_0$				$E_{sp}g_1$	$E_{sp}g_2$	$E_{sp}g_3$
s^a	$E_{ss}g_0$	E_s^a	$-E_{sp}g_1^*$	$-E_{sp}g_2^*$	$-E_{sp}g_3^*$			
p_x^c		$-E_{sp}g_1$	E_p^c			$E_{xx}g_0$	$E_{xy}g_3$	$E_{xy}g_2$
p_y^c		$-E_{sp}g_2$		E_p^c		$E_{xy}g_3$	$E_{xx}g_0$	$E_{xy}g_1$
p_z^c		$-E_{sp}g_3$			E_p^a	$E_{xy}g_2$	$E_{xy}g_1$	$E_{xx}g_0$
p_x^a	$E_{sp}g_1^*$		$E_{xx}g_0^*$	$E_{xy}g_3^*$	$E_{xy}g_2^*$	E_p^a		
p_y^a	$E_{sp}g_2^*$		$E_{xy}g_3^*$	$E_{xx}g_0^*$	$E_{xy}g_1^*$		E_p^a	
p_z^a	$E_{sp}g_3^*$		$E_{xy}g_2^*$	$E_{xy}g_1^*$	$E_{xx}g_0^*$			E_p^a

© 2000 Richard P. Muller

Harrison Parameters

- C and A refer to *cation* and *anion*
  - Ga,N for a 3,5 semiconductor in Cubic Zincblende form
  - Si,Si for a pure semiconductor in Diamond form
- E_s, E_{sp}, E_{xx}, E_{xy} are fit to experiment
- g_0, g_1, g_2, g_3 are functions of the *k-vectors*
  - k-vectors are phase factors in reciprocal space
  - show how band various in different space group symmetries
  - $g_0(\mathbf{k}) = e^{-i\mathbf{kd1}} + e^{-i\mathbf{kd2}} + e^{-i\mathbf{kd3}} + e^{-i\mathbf{kd4}}$
  - Tetrahedral directions: $d_1 = [111]a/4$, $d_2 = [1\text{-}1\text{-}1]a/4$

© 2000 Richard P. Muller

Outline of TB Program

```
kpoints = get_k_points(N)
energy_archive = []
for kpoint in kpoints:
  H = get_TB_Hamiltonian(kpoint)
  energies = eigenvalues(H)
  energies = ev_sort(energies)
  energy_archive.append(energies)
gnuplot_output(energy_archive)
```

© 2000 Richard P. Muller

get_k_points function

```
def get_k_points(N):
  kpoints = []
  kx,ky,kz = 0.5,0.5,0.5 #L Point
  k_points.append((kx,ky,kz))
  step = 0.5/float(N)
  for i in range(N):  # Move to Gamma (0,0,0)
       kx,ky,kz = kx-step,ky-step,kz-step
       k_points.append((kx,ky,kz))
  # Similar steps for X (1,0,0) & K (1,1,0)
  return kpoints
```

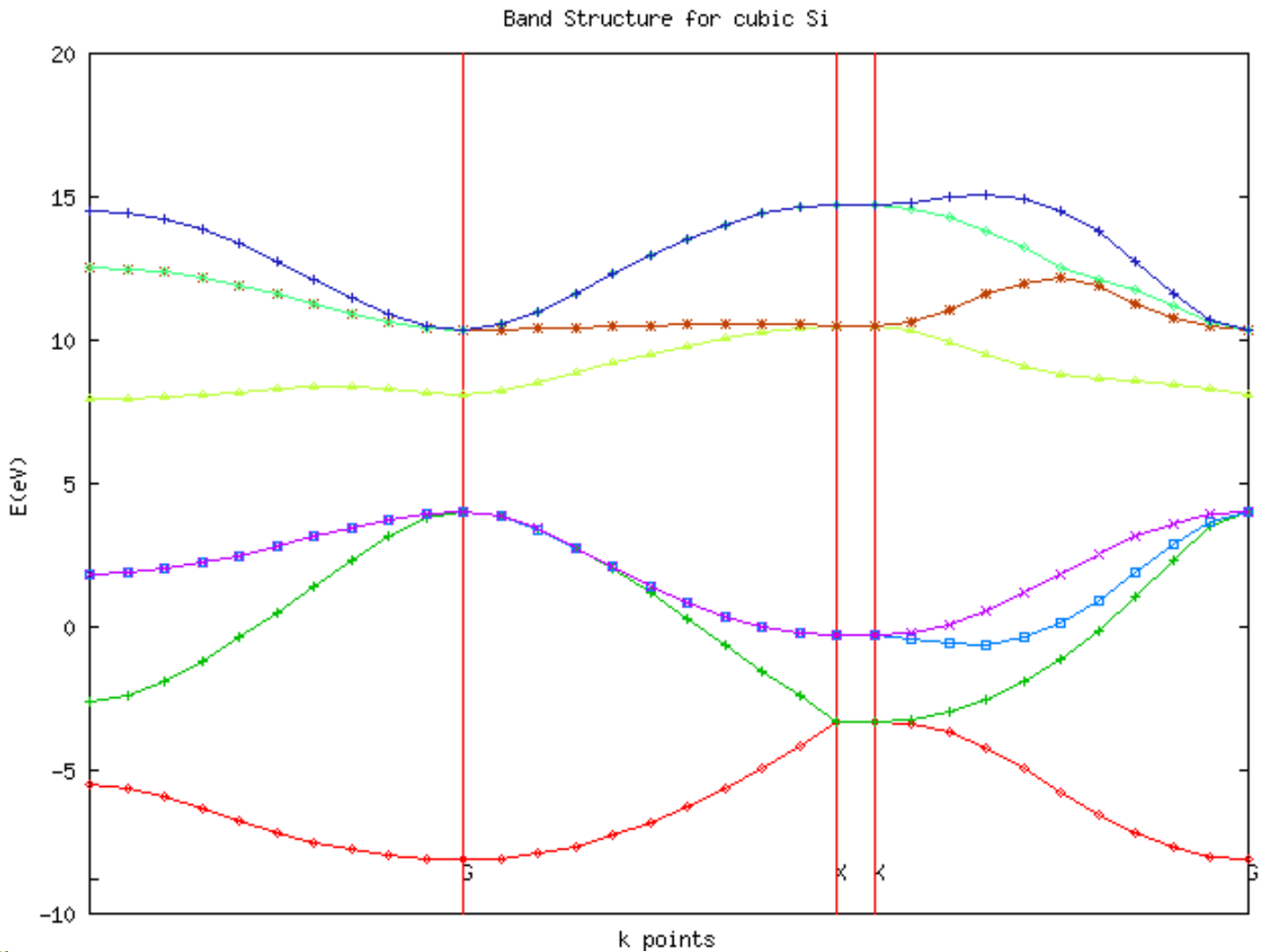
© 2000 Richard P. Muller

get_TB_Hamiltonian function

```
def get_TB_Hamiltonian(kpoint):
  phase_factors = get_phases(kpoint)
  H = zeros((8,8),Complex)
  H = set_diag_values(H)
  H = set_off_diag_values(H,phase_factors)
  return H
```

© 2000 Richard P. Muller

Band Structure Output



© 2000 Richard P. Muller

Numeric Python References

- http://numpy.sourceforge.net NumPy Web Site
- http://numpy.sourceforge.net/numpy.pdf NumPy Documentation
- http://starship.python.net/crew/hinsen/scientific.html Konrad Hinsen's Scientific Python page, a set of Python modules useful for scientists, including the LeastSquares package.
- http://starship.python.net/crew/hinsen/MMTK/ Konrad Hinsen's Molecular Modeling Tool Kit, a biological molecular modeling kit written using Numerical Python.
- http://oliphant.netpedia.net/ Travis Oliphant's Python Pages, including: FFTW, Sparse Matrices, Special Functions, Signal Processing, Gaussian Quadrature, Binary File I/O

© 2000 Richard P. Muller

Tight Binding References

W. A. Harrison. *Electronic Structure and the Properties of Solids.* Dover (New York, 1989).

D. J. Chadi and M. L. Cohen. "Tight Binding Calculations of the Valence Bands of Diamond and Zincblende Crystals." Phys. Stat. Solids. **B68,** 405 (1975).

http://www.wag.caltech.edu/home/rpm/projects/tight-binding/

My tightbinding programs. Includes one that reproduces Harrison's method, and one that reproduces Chadi and Cohen's methods (the parameterization differs slightly).

http://www.wag.caltech.edu/home/rpm/python_course/tb.py

Simplified (and organized) TB program.

© 2000 Richard P. Muller