

Special Topics in Computational Physics

Special Topics in Computational Physics:

- Preliminary Setup and Configuration
- QuTiP: Quantum Toolbox in Python
- Partial Differential Equations: Laplace and Wave equations

Preliminary Setup and Configuration

Download and install anaconda from <https://anaconda.com/download>

The screenshot shows the Anaconda Distribution website. At the top, there is a dark header with the Anaconda logo, navigation links for Enterprise, Pricing, Resources, and About, and two buttons: "Sign In" and "Contact Sales". Below the header, the page title "Anaconda Distribution" is displayed in green. The main feature is a large, bold "Free Download" button. Below it, a sub-headline reads "Everything you need to get started in data science on your workstation." To the right of the text, there is a decorative graphic of green wavy lines. A bulleted list of features follows: "✓ Free distribution install", "✓ Thousands of the most fundamental DS, AI, and ML packages", "✓ Manage packages and environments from desktop application", and "✓ Deploy across hardware and software platforms". At the bottom left, there is a "Download" button with a download icon, and below it, links for "Get Additional Installers" and icons for Windows, Mac, and Linux.

ANACONDA.

Enterprise Pricing Resources About

Sign In Contact Sales

Anaconda Distribution

Free Download

Everything you need to get started in data science on your workstation.

- ✓ Free distribution install
- ✓ Thousands of the most fundamental DS, AI, and ML packages
- ✓ Manage packages and environments from desktop application
- ✓ Deploy across hardware and software platforms

Download

Get Additional Installers

Windows | Mac | Linux

Preliminary Setup and Configuration

Download and install anaconda from <https://anaconda.com/download>

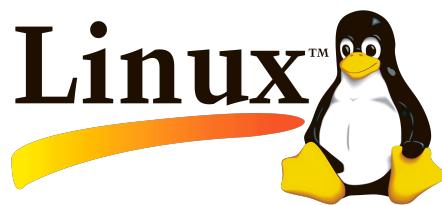
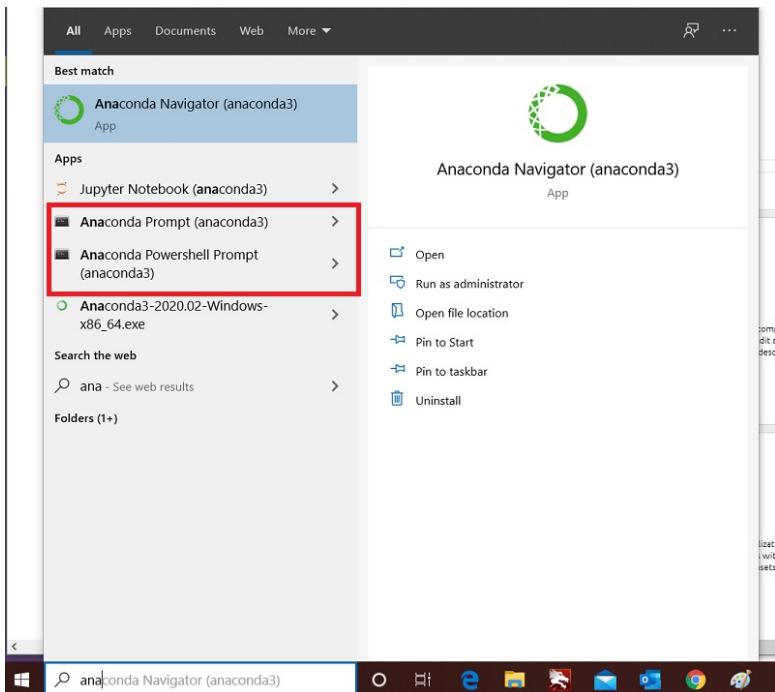
The screenshot shows the Anaconda Distribution website. At the top, there is a dark header with the "ANACONDA." logo, navigation links for "Enterprise", "Pricing", "Resources", and "About", and two buttons: "Sign In" and "Contact Sales". Below the header, the text "Anaconda Distribution" is displayed in green, followed by a large white "Free Download" button. A descriptive paragraph states, "Everything you need to get started in data science on your workstation." To the left of this text is a bulleted list of four features: "Free distribution install", "Thousands of the most fundamental DS, AI, and ML packages", "Manage packages and environments from desktop application", and "Deploy across hardware and software platforms". At the bottom left, there is a blue oval containing a "Download" button with a green icon, and below it, links for "Get Additional Installers" and platform icons for Windows, Mac, and Linux.

Preliminary Setup and Configuration

Load a ‘conda shell’ with the base environment activated.



Windows

A screenshot of a terminal window on an Ubuntu system. The window title is "daneel@sudarshan:~". The terminal prompt shows the user has run the command "conda activate base" and is now in the "anaconda3" environment. The background of the window is dark purple.

Preliminary Setup and Configuration

- Use the conda package manager to add the ‘conda-forge’ repository
- After loading the conda shell, create a special conda environment named ‘exercises’, together with the required packages

```
$ conda config --add channels conda-forge  
$ conda create -n exercises python=3.11 numpy scipy matplotlib qutip
```

Note that:

- ¹ The conda dependency resolver is a bit slow, so this can take some time to finish.
- ² This environment is named ‘exercises’, but you can use any name that you want.
- ³ This environment runs a more recent version of python (version 3.11) than the anaconda default.

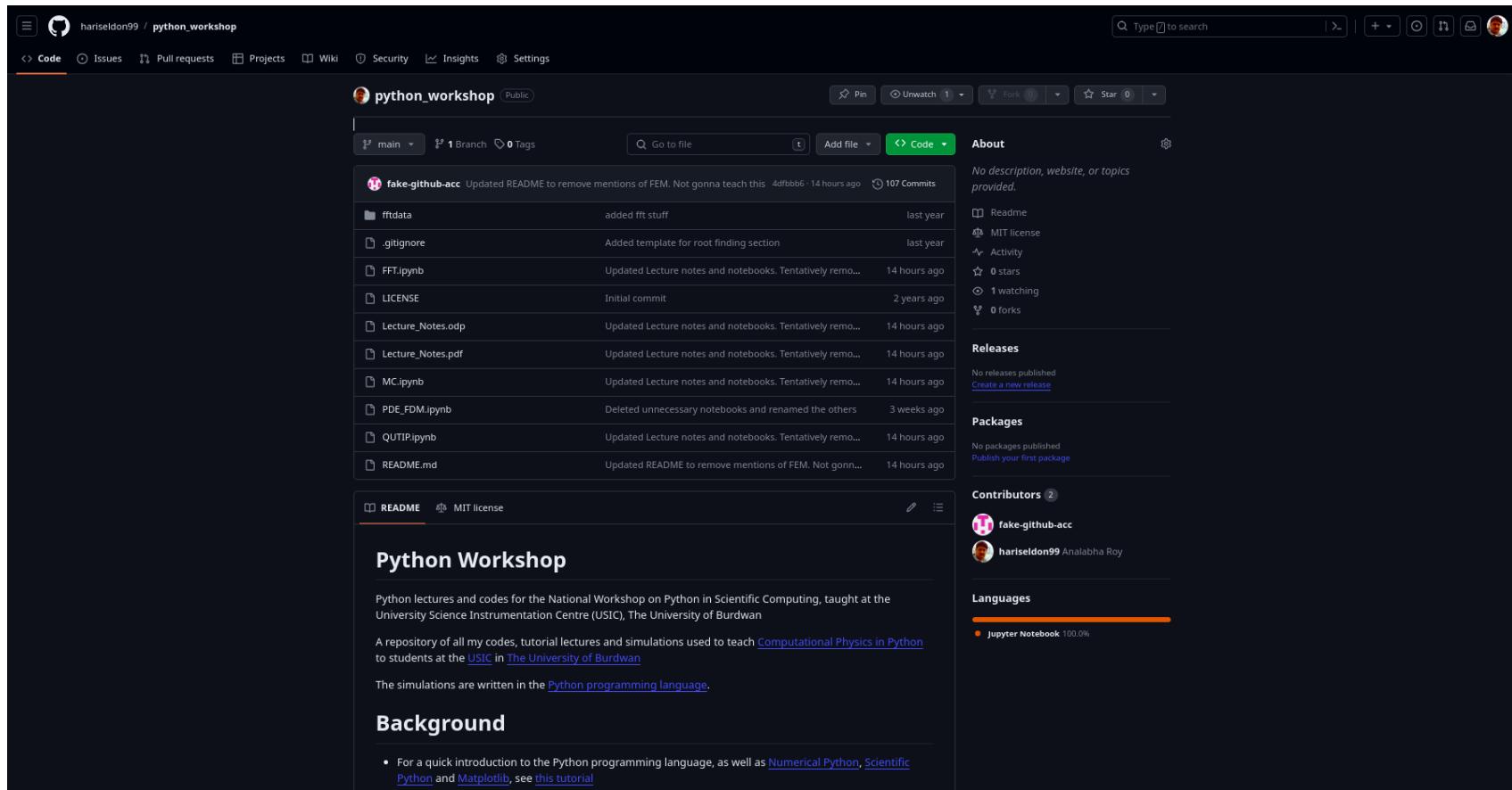
This is suggested because python versions 3.11 and higher perform better than the default version.

- Activate this environment whenever you want to use it by loading the conda shell and running

```
$ conda activate exercises  
$ jupyter notebook
```

Preliminary Setup and Configuration

- All materials for this workshop session are up on my GitHub Repository.
- To view, use a web browser to visit
https://github.com/hariseldon99/python_workshop



The screenshot shows the GitHub repository page for 'python_workshop'. The repository is public and has 107 commits. The file tree shows several Jupyter notebooks (ipynb files) and a README file. The README contains a brief description of the Python Workshop, mentioning it's for the National Workshop on Python in Scientific Computing at the University Science Instrumentation Centre (USIC), The University of Burdwan. It also states that it's a repository of codes, tutorial lectures, and simulations for teaching Computational Physics in Python. The Background section lists resources for learning Python programming, Numerical Python, Scientific Python, and Matplotlib. The repository has 1 star, 1 watch, and 0 forks. Contributors listed are fake-github-acc and hariseldon99 (Analabha Roy). The repository uses Jupyter Notebook as its primary language.

Code Issues Pull requests Projects Wiki Security Insights Settings

python_workshop (Public)

fake-github-acc Updated README to remove mentions of FEM. Not gonna teach this 4dfbbb6 · 14 hours ago 107 Commits

fftdata added fft stuff last year

.gitignore Added template for root finding section last year

FFT.ipynb Updated Lecture notes and notebooks. Tentatively remo... 14 hours ago

LICENSE Initial commit 2 years ago

Lecture_Notes.odp Updated Lecture notes and notebooks. Tentatively remo... 14 hours ago

Lecture_Notes.pdf Updated Lecture notes and notebooks. Tentatively remo... 14 hours ago

MC.ipynb Updated Lecture notes and notebooks. Tentatively remo... 14 hours ago

PDE_FDM.ipynb Deleted unnecessary notebooks and renamed the others 3 weeks ago

QUTIP.ipynb Updated Lecture notes and notebooks. Tentatively remo... 14 hours ago

README.md Updated README to remove mentions of FEM. Not gonn... 14 hours ago

README MIT license

Python Workshop

Python lectures and codes for the National Workshop on Python in Scientific Computing, taught at the University Science Instrumentation Centre (USIC), The University of Burdwan

A repository of all my codes, tutorial lectures and simulations used to teach [Computational Physics in Python](#) to students at the [USIC](#) in [The University of Burdwan](#)

The simulations are written in the [Python programming language](#).

Background

- For a quick introduction to the Python programming language, as well as [Numerical Python](#), [Scientific Python](#) and [Matplotlib](#), see [this tutorial](#)

About

No description, website, or topics provided.

Readme

MIT license

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Contributors 2

fake-github-acc

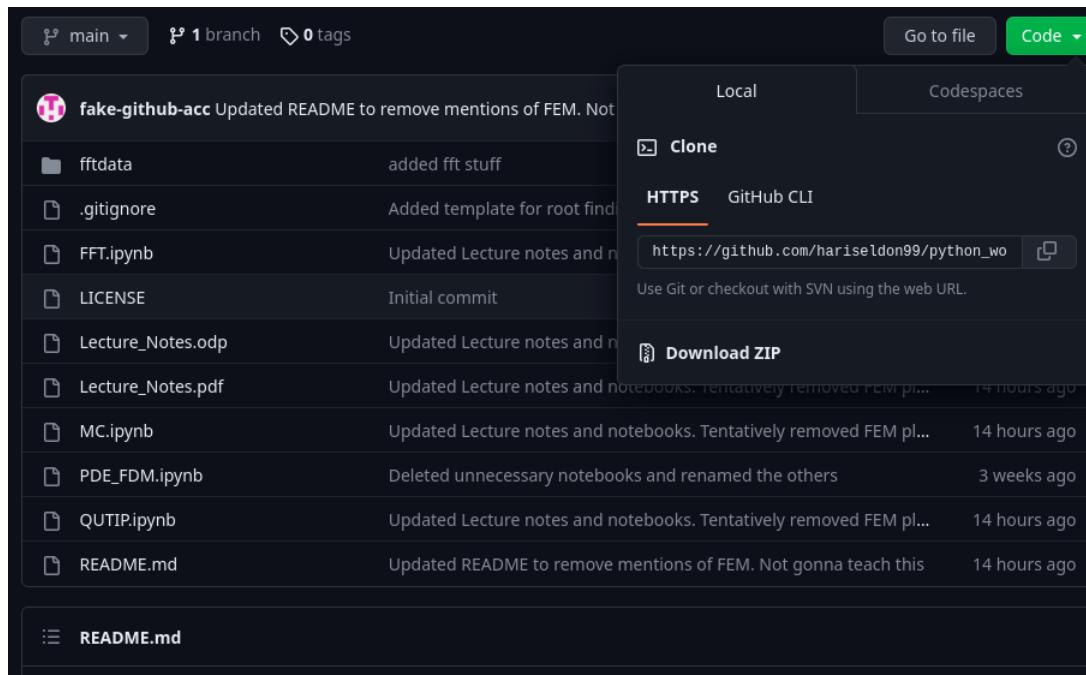
hariseldon99 Analabha Roy

Languages

Jupyter Notebook 100.0%

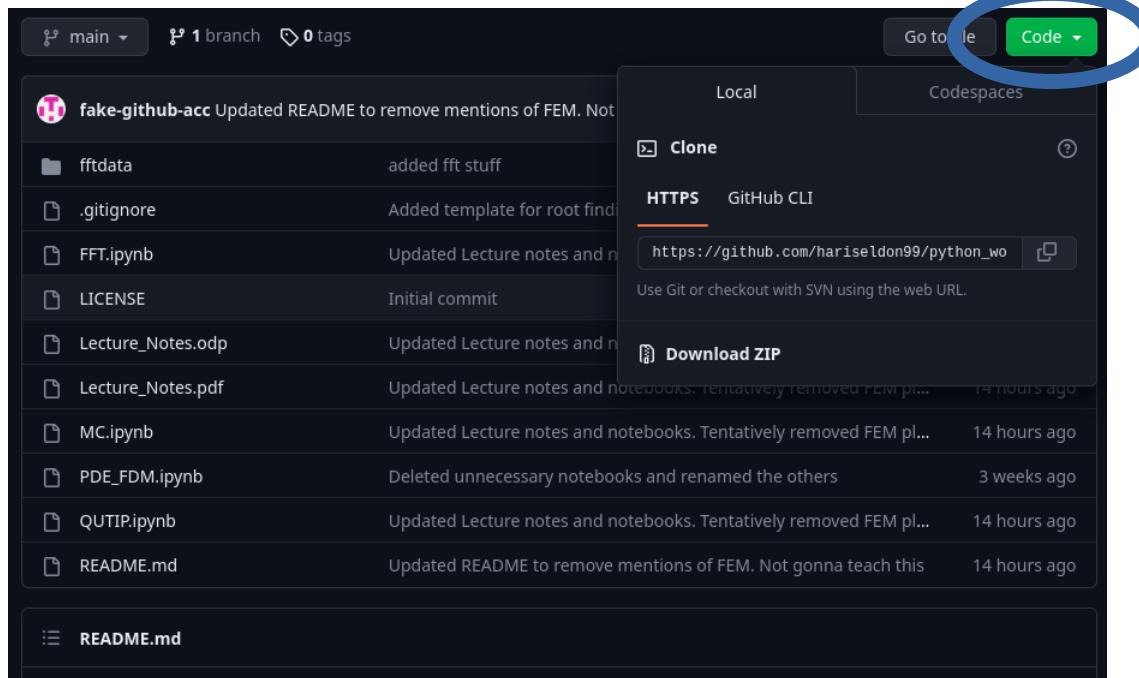
Preliminary Setup and Configuration

- All materials for this workshop session are up on my GitHub Repository.
- To view, use a web browser to visit
https://github.com/hariseldon99/python_workshop
- To download this repository on your local machine, click on  above the list of files, then click ‘Download ZIP’ in the dropdown menu



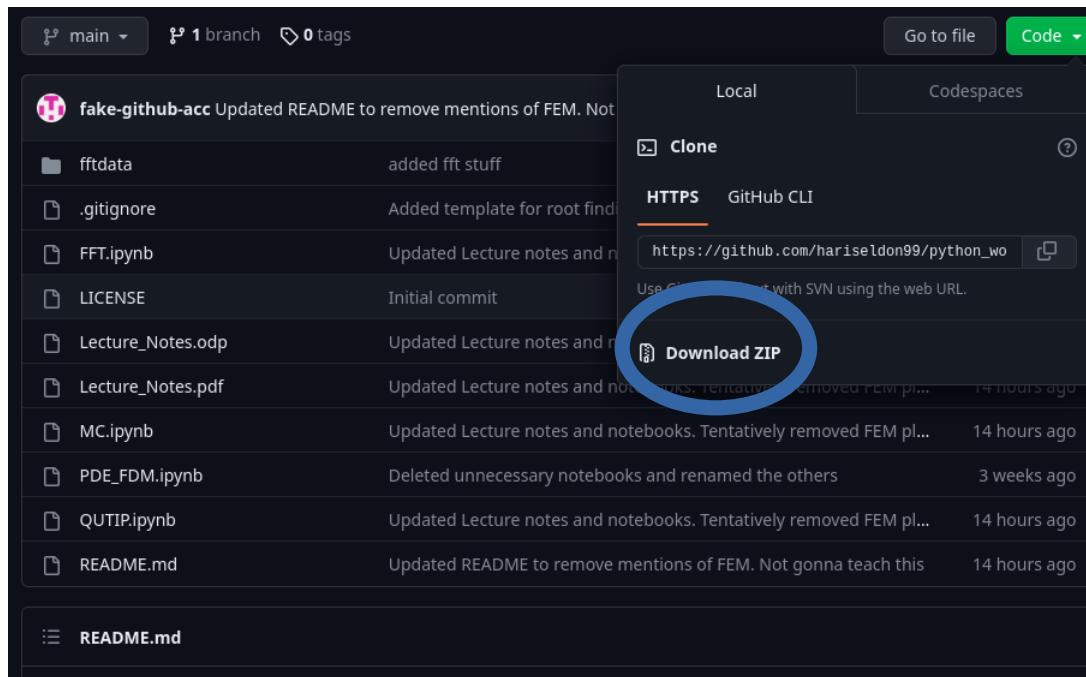
Preliminary Setup and Configuration

- All materials for this workshop session are up on my GitHub Repository.
- To view, use a web browser to visit
https://github.com/hariseldon99/python_workshop
- To download this repository on your local machine, click on  above the list of files, then click ‘Download ZIP’ in the dropdown menu



Preliminary Setup and Configuration

- All materials for this workshop session are up on my GitHub Repository.
- To view, use a web browser to visit
https://github.com/hariseldon99/python_workshop
- To download this repository on your local machine, click on  above the list of files, then click ‘Download ZIP’ in the dropdown menu



QuTiP: The Quantum Toolbox in Python

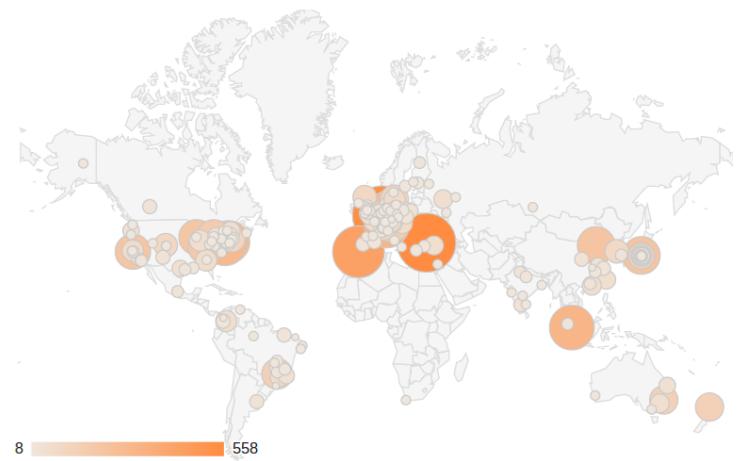
QuTiP: The Quantum Toolbox in Python

- Framework for computational quantum dynamics
 - ~ Efficient and easy to use for quantum physicists
 - ~ Thoroughly tested (100+ unit tests)
 - ~ Well documented (200+ pages, 50+ examples)
 - ~ Quite large number of users (>1000 downloads)
- Suitable for
 - ~ theoretical modeling and simulations
 - ~ modeling experiments
- 100% open source
- Implemented in Python/Cython using SciPy, Numpy, and matplotlib

QuTiP: The Quantum Toolbox in Python

Project information

- Authors: Paul Nation and Robert Johansson
- Web site: <http://qutip.org>
- License: GPLv3
- Repository: <http://github.com/qutip>
- Publication: Comp. Phys. Comm. **183**, 1760 (2012)
arXiv:1211.6518 (2012)



QuTiP: The Quantum Toolbox in Python

- **Objectives**

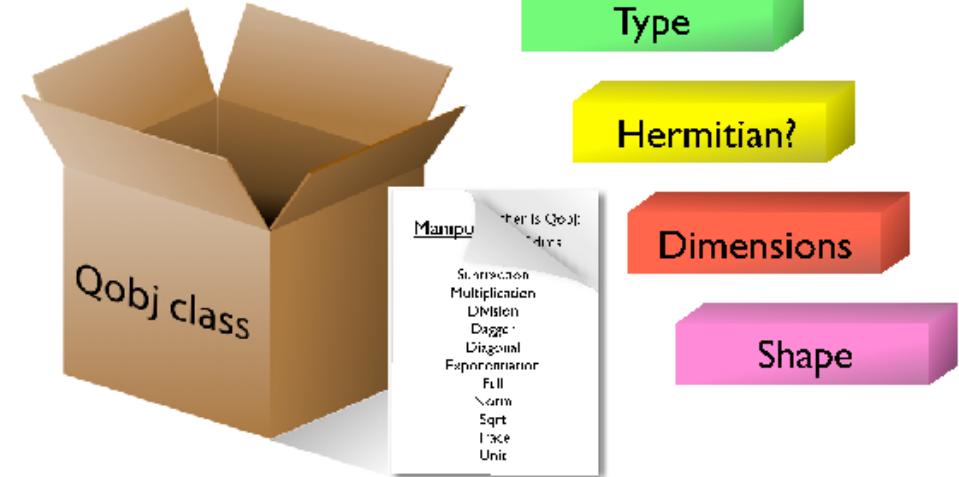
To provide a powerful framework for quantum mechanics that closely resembles the standard mathematical formulation

- ~ Efficient and easy to use
- ~ General framework, able to handle a wide range of different problems

- **Design and implementation**

- ~ Object-oriented design
- ~ Qobj class used to represent quantum objects
 - Operators
 - State vectors
 - Density matrices
- ~ Library of utility functions that operate on Qobj instances

QuTiP core class:
Qobj



QuTiP: The Quantum Toolbox in Python

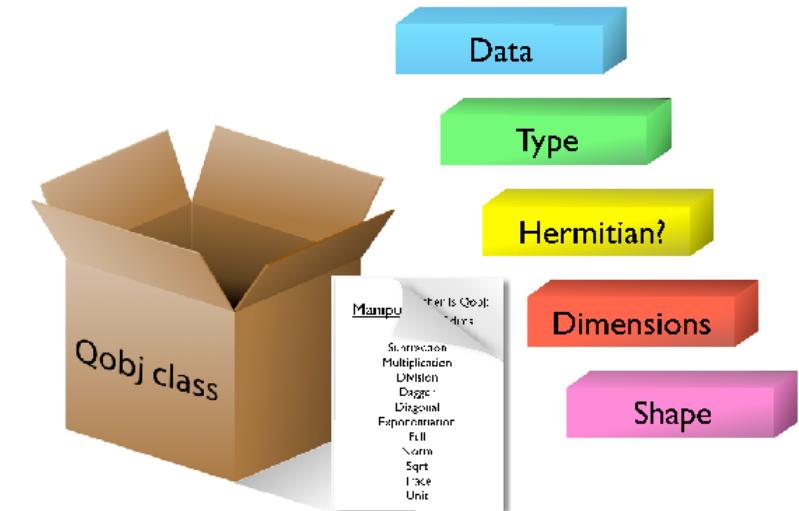
Quantum object class: Qobj

- Abstract representation of quantum states and operators
 - ~ Matrix representation of the object
 - ~ Structure of the underlying state space, Hermiticity, type, etc.
 - ~ Methods for performing all common operations on quantum objects:
`eigs(), dag(), norm(), unit(), expm(), sqrt(), tr(), ...`
 - ~ Operator arithmetic with implementations of: +, -, *, ...

Example: built-in operator $\hat{\sigma}_x$

```
>>> sigmax()

Quantum object: dims = [[2], [2]], shape = [2, 2],
type = oper, isHerm = True
Qobj data =
[[ 0.  1.]
 [ 1.  0.]]
```



Example: built-in state $|\alpha = 0.5\rangle$

```
>>> coherent(5, 0.5)

Quantum object: dims = [[5], [1]], shape = [5, 1], type = ket
Qobj data =
[[ 0.88249693]
 [ 0.44124785]
 [ 0.15601245]
 [ 0.04496584]
 [ 0.01173405]]
```

QuTiP: The Quantum Toolbox in Python

Evolution of quantum systems

The main use of QuTiP is quantum evolution. A number of solvers are available.

- Typical simulation workflow:
 - i. Define parameters that characterize the system
 - ii. Create Qobj instances for operators and states
 - iii. Create Hamiltonian, initial state and collapse operators, if any
 - iv. Choose a solver and evolve the system
 - v. Post-process, visualize the data, etc.
- Available evolution solvers:
 - ~ Unitary evolution: Schrödinger and von Neumann equations
 - ~ Lindblad master equations
 - ~ Monte-Carlo quantum trajectory method
 - ~ Bloch-Redfield master equation
 - ~ Floquet-Markov master equation
 - ~ Propagators

QuTiP: The Quantum Toolbox in Python

Lindblad master equation

Equation of motion for the density matrix $\rho(t)$ for a quantum system that interacts with its environment:

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t), \rho(t)] + \sum_n \frac{1}{2} [2c_n\rho(t)c_n^\dagger - \rho(t)c_n^\dagger c_n - c_n^\dagger c_n \rho(t)]$$

$H(t)$ = system Hamiltonian

$c_n = \sqrt{\gamma_n}a_n$ describes the effect of the environment on the system

γ_n = rate of the environment-system interaction process

How do we solve this equation numerically?

- I. Construct the matrix representation of all operators
- II. Evolve the ODEs for the unknown elements in the density matrix
- III. For example, calculate expectation values for some selected operators for each $\rho(t)$

QuTiP: The Quantum Toolbox in Python

Lindblad master equation

Equation of motion for the density matrix $\rho(t)$ for a quantum system that interacts with its environment:

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t), \rho(t)] + \sum_n \frac{1}{2} [2c_n\rho(t)c_n^\dagger - \rho(t)c_n^\dagger c_n - c_n^\dagger c_n \rho(t)]$$

$H(t)$ = system Hamiltonian

$c_n = \sqrt{\gamma_n}a_n$ describes the effect of the environment on the system

γ_n = rate of the environment-system interaction process

How do we solve this equation numerically in QuTiP?

```
from qutip import *

psi0 = ...          # initial state
H   = ...          # system Hamiltonian
c_op_list = [...]  # collapse operators
e_op_list = [...]  # expectation value operators

tlist = linspace(0, 10, 100)
result = mesolve(H, psi0, tlist, c_op_list, e_op_list)
```

QuTiP: The Quantum Toolbox in Python

Example: time-dependence

Multiple Landau-Zener transitions

$$\hat{H}(t) = -\frac{\Delta}{2}\hat{\sigma}_z - \frac{\epsilon}{2}\hat{\sigma}_x - A \cos(\omega t)\hat{\sigma}_z$$

```
from qutip import *

# Parameters
epsilon = 0.0
delta = 1.0

# Initial state: start in ground state
psi0 = basis(2,0)

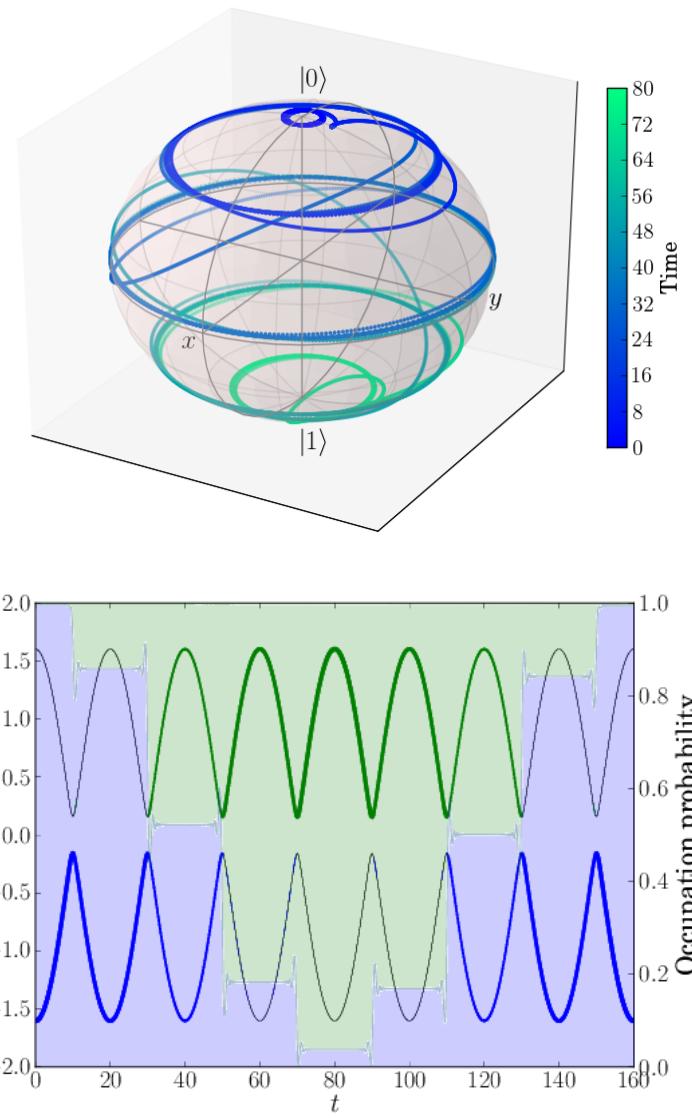
# Hamiltonian
H0 = - delta * sigmaz() - epsilon * sigmax()
H1 = - sigmaz()
h_t = [H0, [H1, 'A * cos(w*t)']]
args = {'A': 10.017, 'w': 0.025*2*pi}

# No dissipation
c_ops = []

# Expectation values
e_ops = [sigmax(), sigmay(), sigmaz()]

# Evolve the system
tlist = linspace(0, 160, 500)
output = mesolve(h_t, psi0, tlist, c_ops, e_ops, args)

# Process and plot result
# ...
```



QuTiP: The Quantum Toolbox in Python

QuTiP: Examples

```
from qutip import *

def evals_multw(w1list, w2, w3, g12, g13):

    # Pre-compute operators for the hamiltonian
    sz1 = tensor(sigmax(), qeye(2), qeye(2))
    sx1 = tensor(sigmax(), qeye(2), qeye(2))

    sz2 = tensor(qeye(2), sigmax(), qeye(2))
    sx2 = tensor(qeye(2), sigmax(), qeye(2))

    sz3 = tensor(qeye(2), qeye(2), sigmax())
    sx3 = tensor(qeye(2), qeye(2), sigmax())

    evals_mat = np.zeros((len(w1list),2**3))
    for idx, w1 in enumerate(w1list):
        # evaluate the Hamiltonian
        H = w1 * sz1 + w2 * sz2 + w3 * sz3 +\
            g12 * sx1 * sx2 + g13 * sx1 * sx3
        # find the energy eigenvalues of the composite system
        evals = H.eigenenergies()
        evals_mat[idx,:] = np.real(evals)

    return evals_mat

w1 = 1.0 * 2 * np.pi # atom 1 frequency: sweep this one
w2 = 0.9 * 2 * np.pi # atom 2 frequency
w3 = 1.1 * 2 * np.pi # atom 3 frequency
g12 = 0.05 * 2 * np.pi # atom1-atom2 coupling strength
g13 = 0.05 * 2 * np.pi # atom1-atom3 coupling strength

w1list = np.linspace(0.75, 1.25, 50) * 2 * np.pi # atom 1 frequency
range
evals_mat = evals_multw(w1list, w2, w3, g12, g13)
```

```
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 20
fig, ax = plt.subplots(figsize=(12,6))

for n in [1,2,3]:
    ax.plot(w1list / (2*np.pi), (evals_mat[:,n]-evals_mat[:,0]) / (2*np.pi), 'b')

ax.set_xlabel('Energy splitting of electron 1')
ax.set_ylabel('Eigenenergies')
ax.set_title('Energy spectrum of three coupled electrons')
plt.show()
```

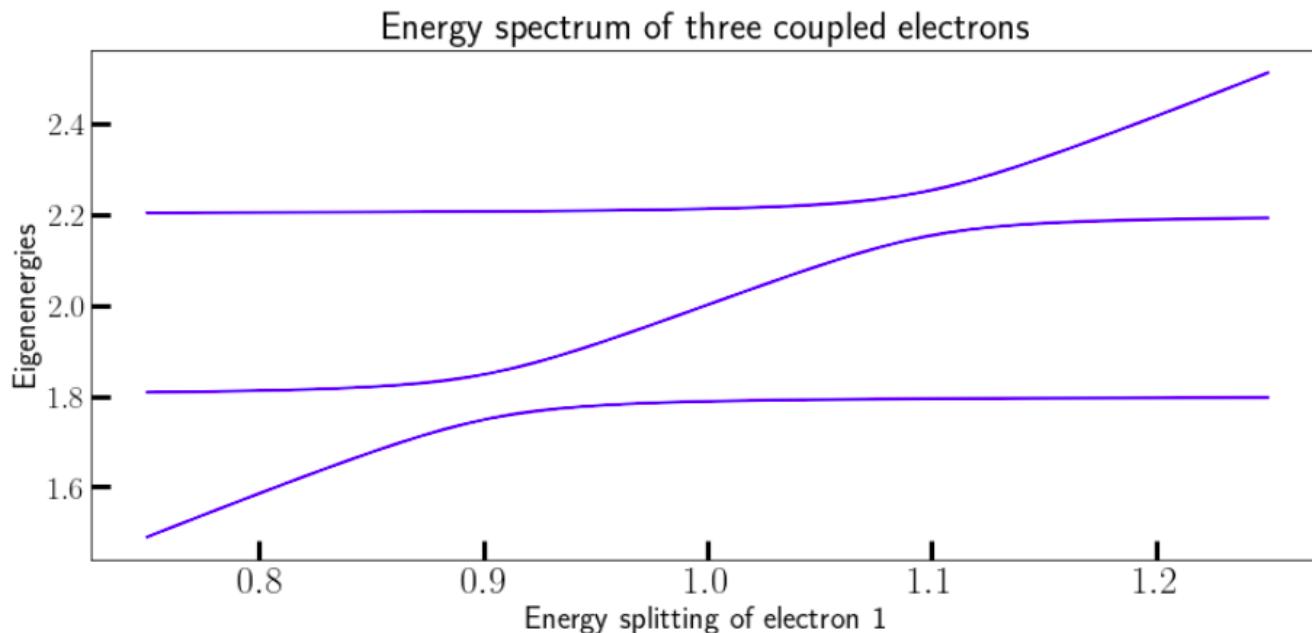
See Jupyter Notebook:
‘QUTIP.ipynb’

$$H = \sum_{i=0}^{N-1} \omega_i \sigma_i^z + g \sum_{i=0}^{N-1} \sigma_i^x \sigma_{i+1}^x$$

QuTiP: The Quantum Toolbox in Python

QuTiP: Examples

$$H = \sum_{i=0}^{N-1} \omega_i \sigma_i^z + g \sum_{i=0}^{N-1} \sigma_i^x \sigma_{i+1}^x$$



See Jupyter Notebook:
‘QUTIP.ipynb’

QuTiP: The Quantum Toolbox in Python

QuTiP: Examples

```
#N = 100
N = 30

H = sum([basis(N,i) * basis(N, i + 1).dag() for i in range(N - 1)]) +\
    sum([basis(N,i) * basis(N, i - 1).dag() for i in range(1, N)])

evals = H.eigenenergies()
# satisfy periodic boundary conditions we need E(-k) = E(k)
numerical = np.concatenate((np.flip(evals, 0), evals), axis=0)

import numpy as np
import matplotlib.pyplot as plt

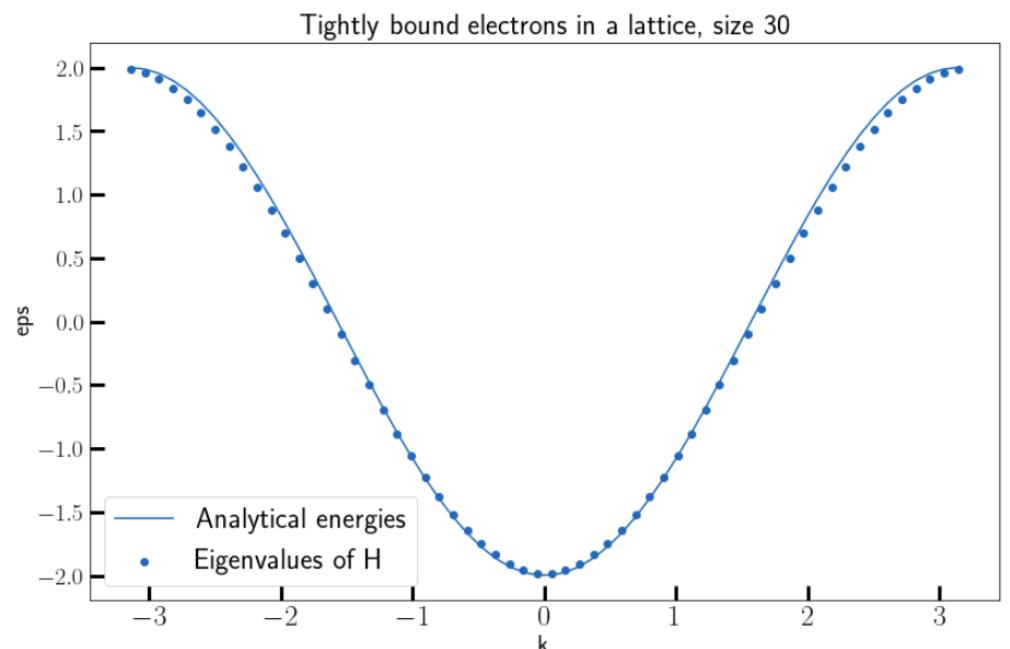
plt.rcParams['figure.figsize'] = (12,8)
plt.rcParams['font.size'] = 20
k = np.linspace(-np.pi, np.pi, 2*N)

# plt.scatter(k[::3],numerical[::3].real, label='Eigenvalues of H')
# plt.scatter(k,numerical.real, label='Eigenvalues of H')

analytical = -2 * np.cos(k)
plt.title(f"Tightly bound electrons in a lattice, size {N}")
plt.plot(k,analytical, label='Analytical energies')
plt.xlabel("k")
plt.ylabel("eps")
plt.legend()
plt.show()
```

See Jupyter Notebook:
‘QUTIP.ipynb’

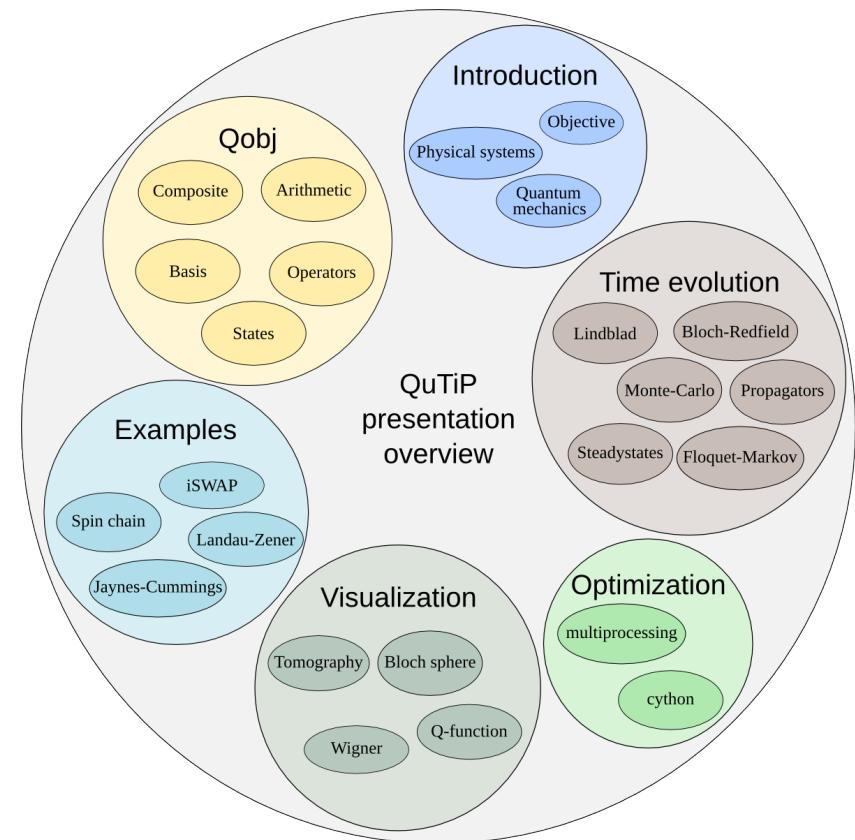
$$H = \frac{1}{2} \sum_{i=0}^{N-1} \left(c_i^\dagger c_{i+1} + c_{i+1}^\dagger c_i \right)$$
$$c_i^\dagger |0\rangle = |i\rangle \quad c|i\rangle = |0\rangle \quad c_i^\dagger |i\rangle = c|0\rangle = 0.$$



QuTiP: The Quantum Toolbox in Python

More information at: <http://qutip.org>

- QuTiP: framework for numerical simulations of quantum systems
 - ~ Generic framework for representing quantum states and operators
 - ~ Large number of dynamics solvers
- Main strengths:
 - ~ Ease of use: complex quantum systems can programmed rapidly and intuitively
 - ~ Flexibility: Can be used to solve a wide variety of problems
 - ~ Performance: Near C-code performance due to use of Cython for time-critical functions



Partial Differential Equations

Partial Differential Equations

PDE: Imposes relations between the various partial derivatives of a multivariable function.

$$f \left(x, y, z \dots, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z} \dots, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \frac{\partial^2 u}{\partial z^2} \dots \right) = 0$$

Solve for $u(x,y,z, \dots)$, subject to **Boundary Conditions**

Partial Differential Equations

Partial Differential Equations

PDE: Imposes relations between the various partial derivatives of a multivariable function.

Types of Linear PDE:

1) Parabolic: $\Delta = 0$ $\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$ Heat Equation

2) Hyperbolic: $\Delta > 0$ $\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$ Wave Equation

3) Elliptic: $\Delta < 0$ $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ Laplace Equation

$$A \frac{\partial^2}{\partial x^2} + 2B \frac{\partial^2}{\partial x \partial y} + C \frac{\partial^2}{\partial y^2} + \dots \text{ lower order terms } \dots = 0$$

$$\Delta \equiv B^2 - AC$$

Partial Differential Equations

Partial Differential Equations

PDE: Imposes relations between the various partial derivatives of a multivariable function.

Types of Boundary Conditions in PDE's:

1) Dirichlet: $u(x, y) \forall (x, y) \in \partial\mathcal{B}$

2) Neumann: $\vec{\nabla}u(x, y) \cdot \hat{n}(x, y) \forall (x, y) \in \partial\mathcal{B}$

3) Cauchy: Dirichlet + Neumann

$$A \frac{\partial^2}{\partial x^2} + 2B \frac{\partial^2}{\partial x \partial y} + C \frac{\partial^2}{\partial y^2} + \dots \text{ lower order terms } \dots = 0$$

Domain \mathcal{B} Boundary $\partial\mathcal{B}$

Partial Differential Equations

Partial Differential Equations

PDE: Imposes relations between the various partial derivatives of a multivariable function.

Numerical Methods for PDE's

Finite Difference Methods (FDM):

Approximate the solutions to using finite difference equations to approximate derivatives. Reduce the ODE to a Linear Algebra Equation

Finite Volume Methods (FVM):

Divide the domain into subdomains and use Gauss/Stoke's theorem to reduce PDE to integral equations, solved using quadratures in algebraic form

Finite Element Methods (FEM):

Divide the domain into subdomains and use Gauss/Stoke's theorem to reduce PDE to integral equations, weighed by a choice of basis states, discretize them solve as linear eqns.

$$A \frac{\partial^2}{\partial x^2} + 2B \frac{\partial^2}{\partial x \partial y} + C \frac{\partial^2}{\partial y^2} + \dots \text{ lower order terms } \dots = 0$$

Domain \mathcal{B} Boundary $\partial\mathcal{B}$

Partial Differential Equations

Partial Differential Equations

PDE: Imposes relations between the various partial derivatives of a multivariable function.

Numerical Methods for PDE's: Unlike ODE's there are no general methods

- Numerical Solutions to PDEs are more complex than ODEs
- Each type of PDE requires its own algorithm
- Algorithms are also dependent on the boundary conditions

$$A \frac{\partial^2}{\partial x^2} + 2B \frac{\partial^2}{\partial x \partial y} + C \frac{\partial^2}{\partial y^2} + \dots \text{ lower order terms } \dots = 0$$

Domain \mathcal{B} Boundary $\partial\mathcal{B}$

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

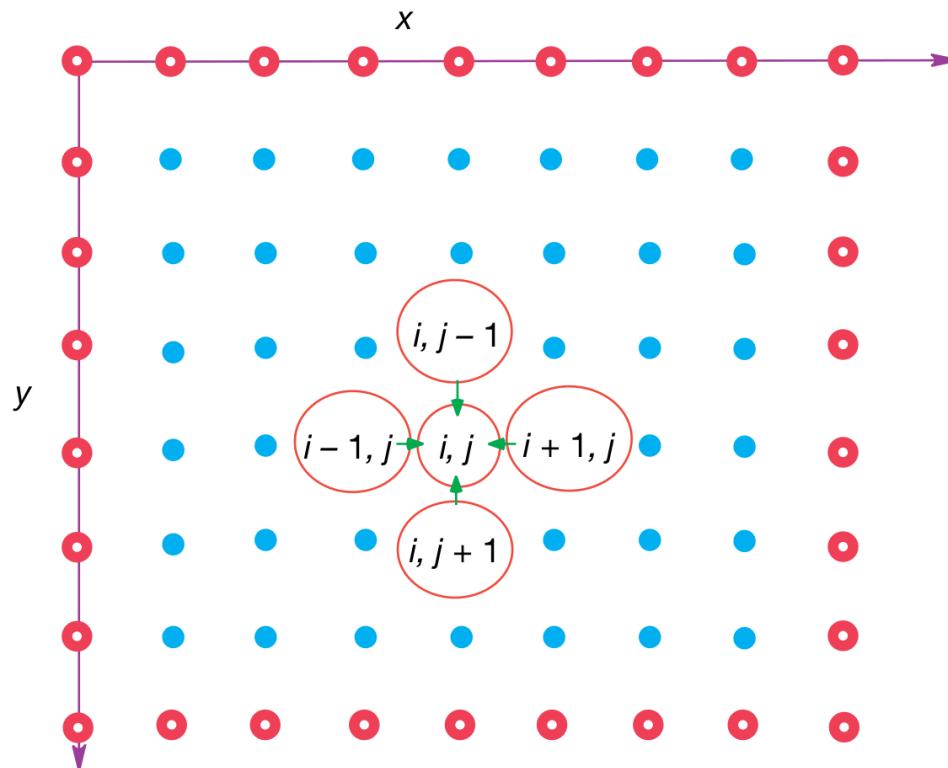
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$

- 1) Divide the 2D Domain into a discrete lattice.
- 2) Replace the partial derivatives by finite difference approximations evaluated at the lattice points
- 3) Reduce the PDE to a system of linear algebra equations, with boundary conditions added as extra equations, then solve them (similar to the ‘matrix method’ for ODE’s)
- 4) Alternatively, reduce the PDE to a system of iterative equations, then update the domain values, starting from the boundary values.

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

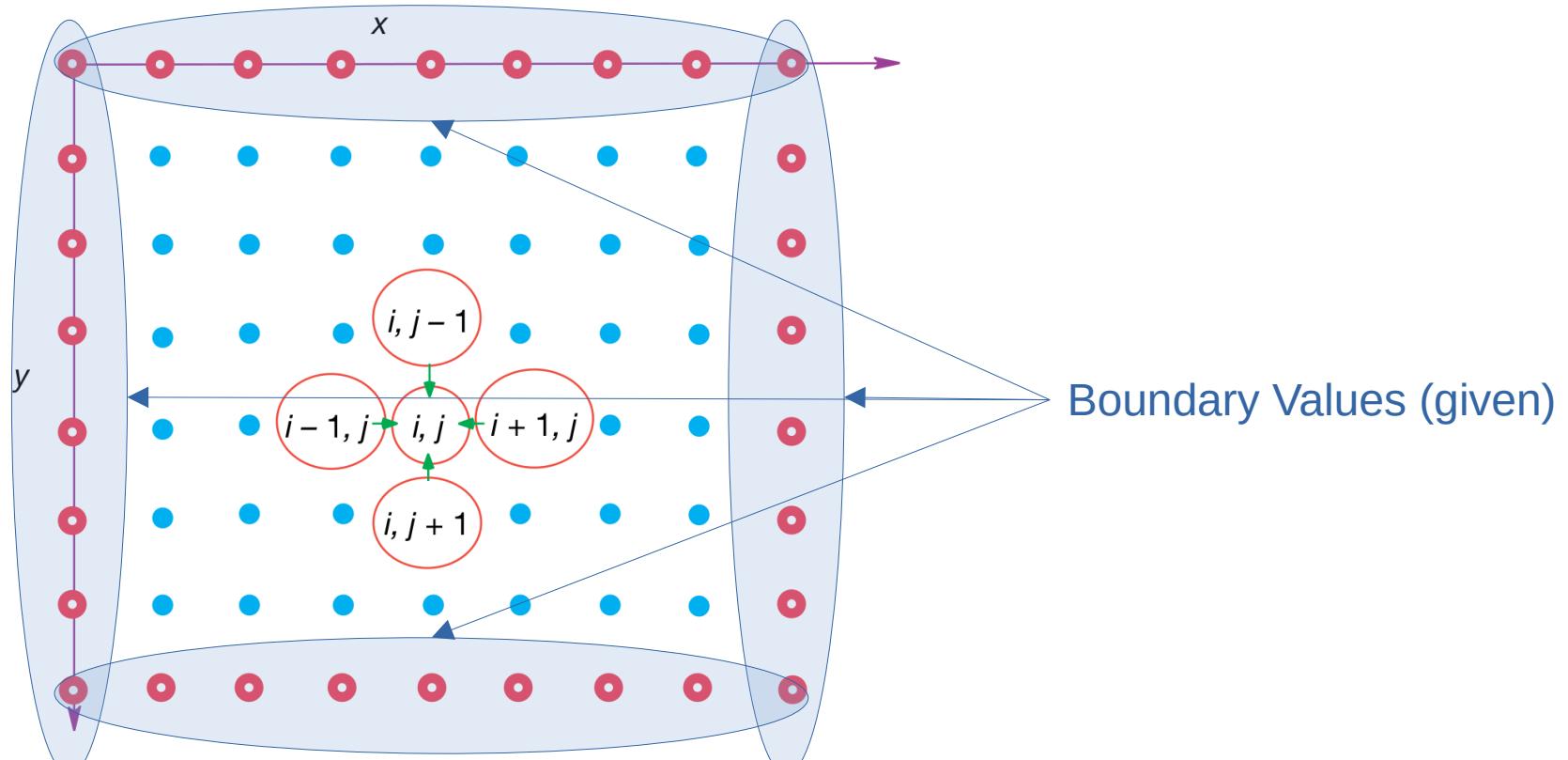
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

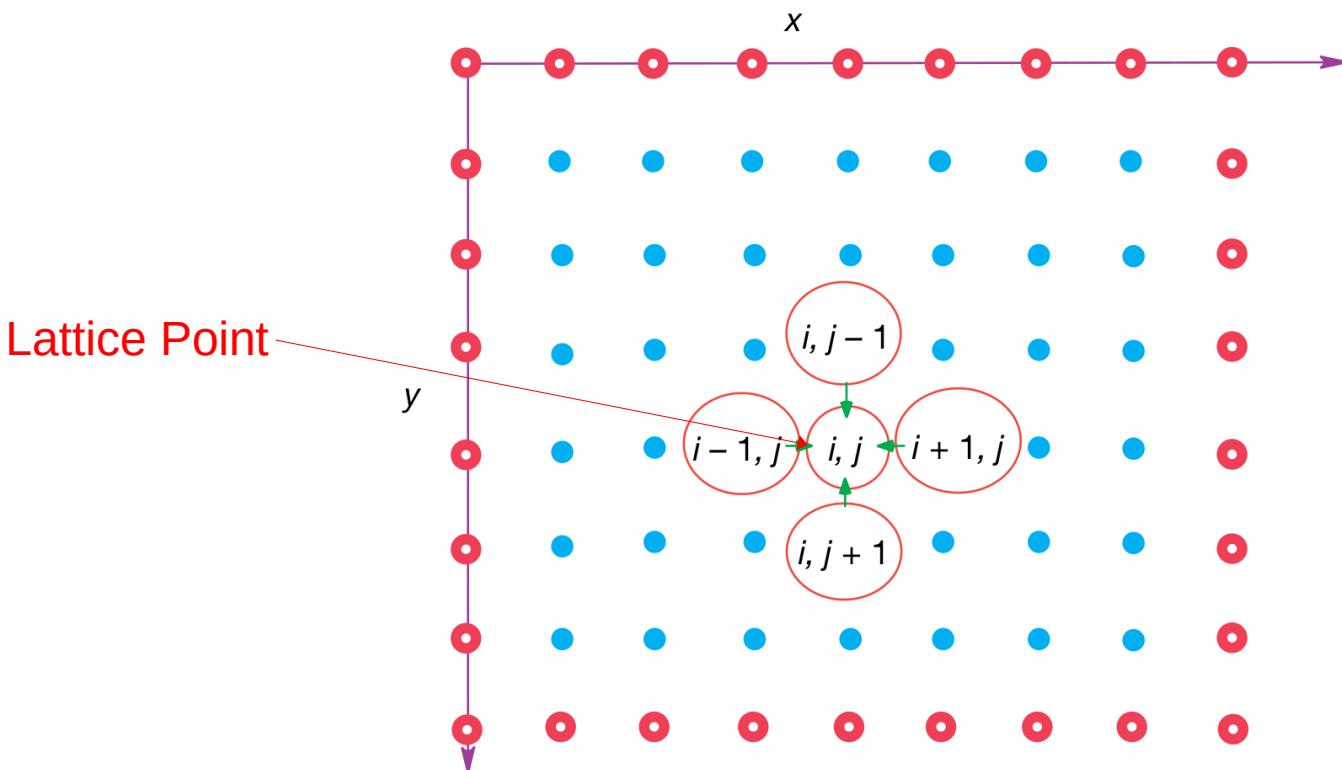
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

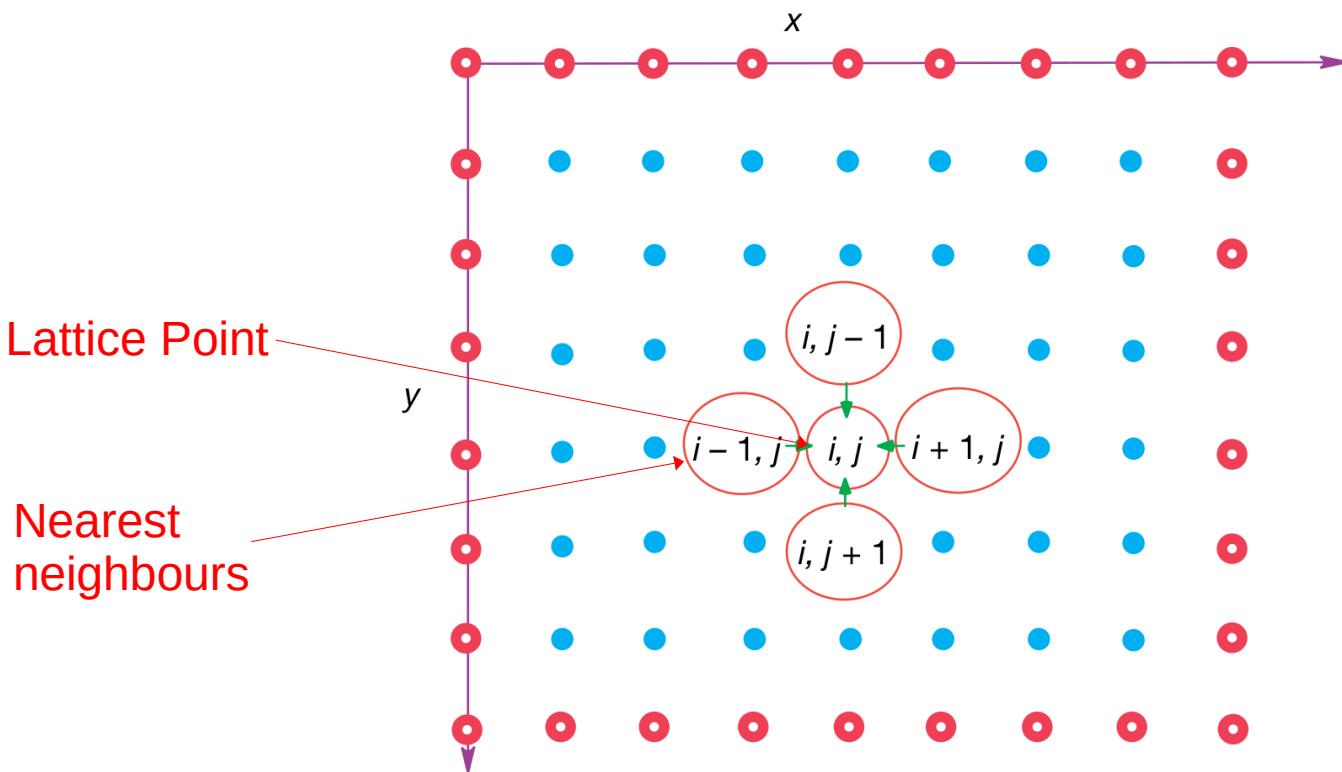
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

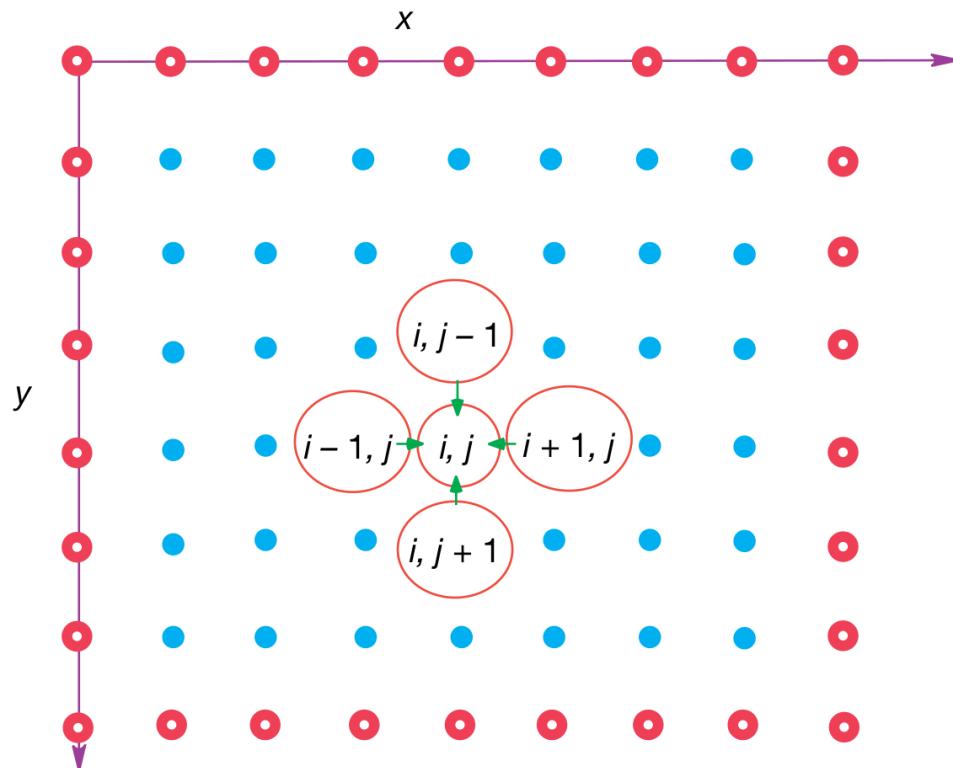
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

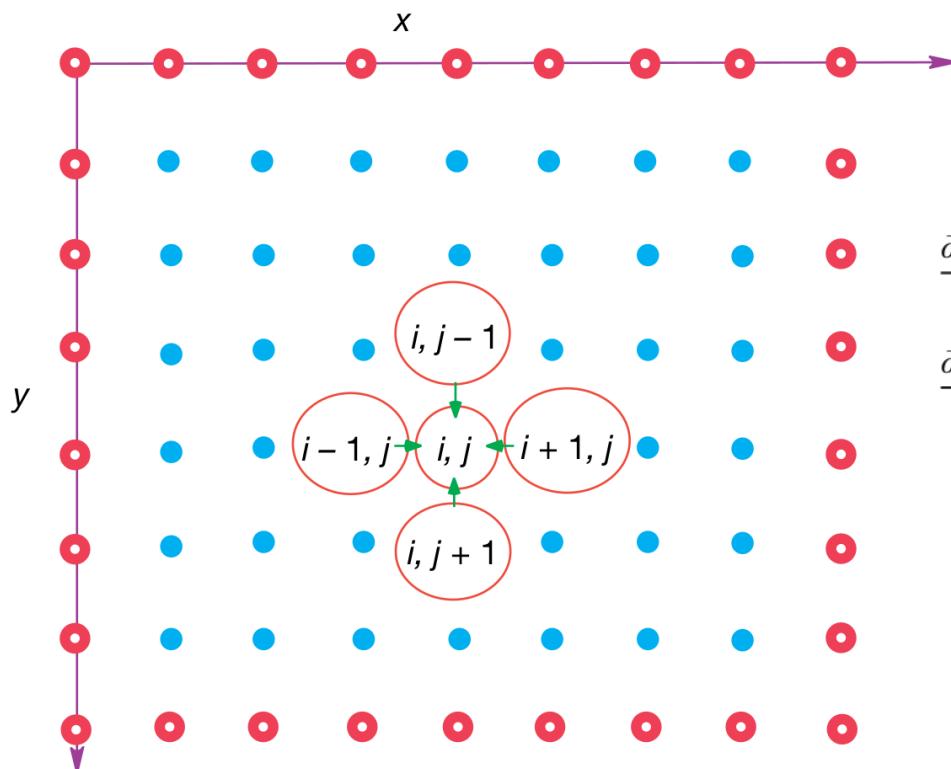
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(x, y)$$



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

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



Three point difference formula!

$$\frac{\partial^2 U(x, y)}{\partial x^2} \simeq \frac{U(x + \Delta x, y) + U(x - \Delta x, y) - 2U(x, y)}{(\Delta x)^2},$$

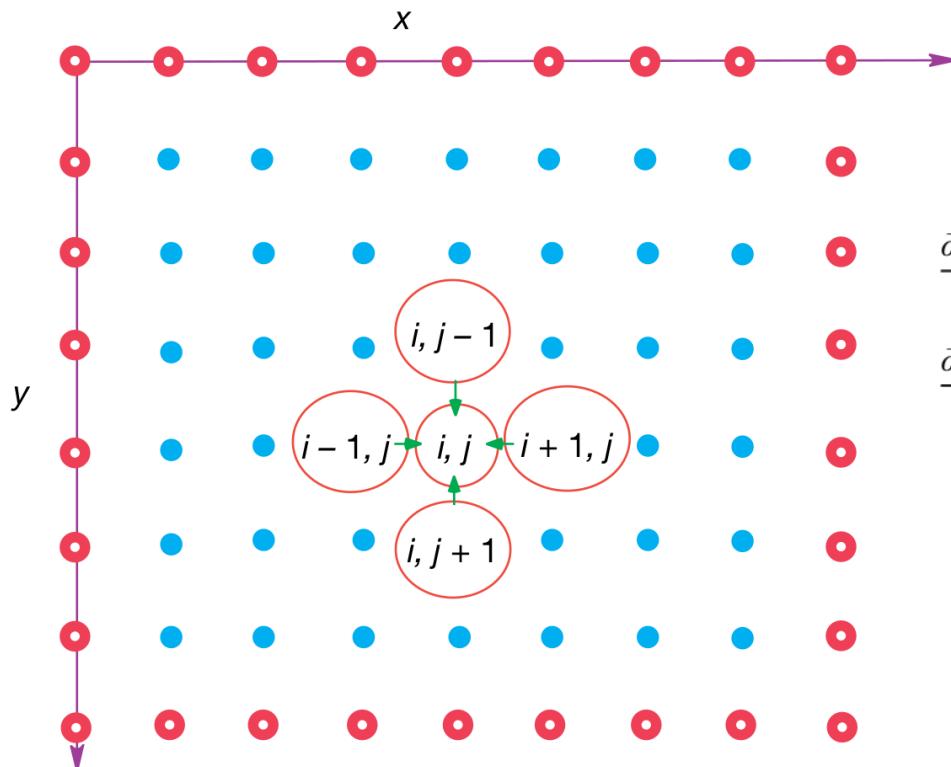
$$\frac{\partial^2 U(x, y)}{\partial y^2} \simeq \frac{U(x, y + \Delta y) + U(x, y - \Delta y) - 2U(x, y)}{(\Delta y)^2}.$$

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

$$\frac{U(x + \Delta x, y) + U(x - \Delta x, y) - 2U(x, y)}{(\Delta x)^2}$$

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



Three point difference formula!

$$\frac{\partial^2 U(x, y)}{\partial x^2} \simeq \frac{U(x + \Delta x, y) + U(x - \Delta x, y) - 2U(x, y)}{(\Delta x)^2} ,$$

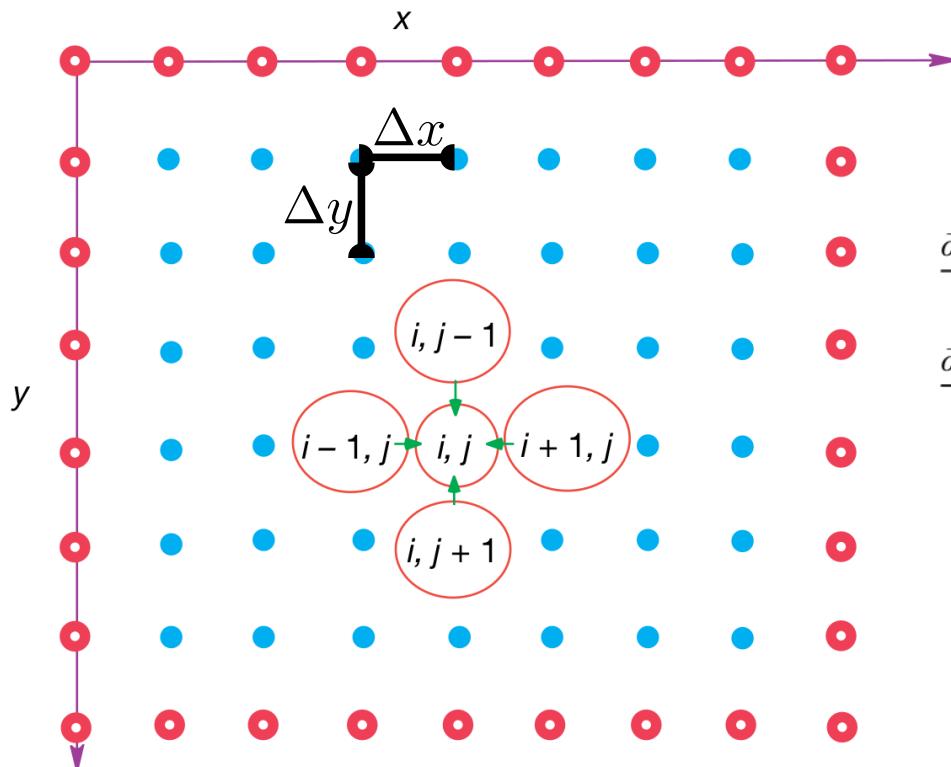
$$\frac{\partial^2 U(x, y)}{\partial y^2} \simeq \frac{U(x, y + \Delta y) + U(x, y - \Delta y) - 2U(x, y)}{(\Delta y)^2} .$$

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

$$\frac{U(x + \Delta x, y) + U(x - \Delta x, y) - 2U(x, y)}{(\Delta x)^2}$$

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



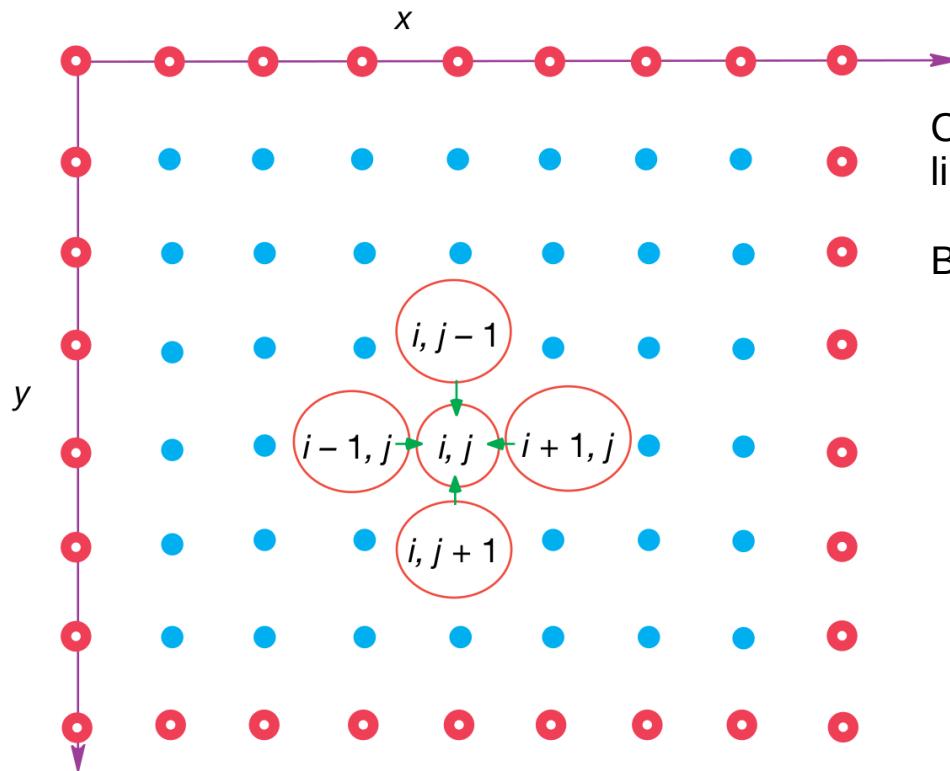
Three point difference formula!

$$\frac{\partial^2 U(x, y)}{\partial x^2} \simeq \frac{U(x + \Delta x, y) + U(x - \Delta x, y) - 2U(x, y)}{(\Delta x)^2} ,$$

$$\frac{\partial^2 U(x, y)}{\partial y^2} \simeq \frac{U(x, y + \Delta y) + U(x, y - \Delta y) - 2U(x, y)}{(\Delta y)^2} .$$

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



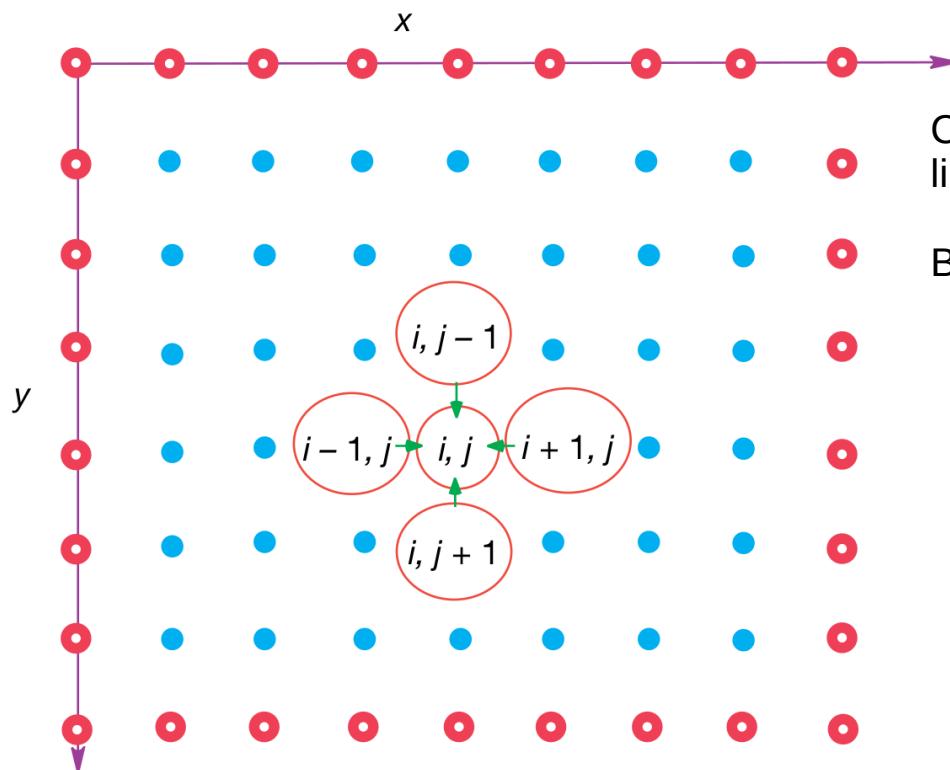
Can solve these equations numerically using linear algebra methods

But, there is a better way (less memory)

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

$$U_{i,j} = \frac{1}{4} [U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}] + \pi\rho(i\Delta, j\Delta)\Delta^2$$



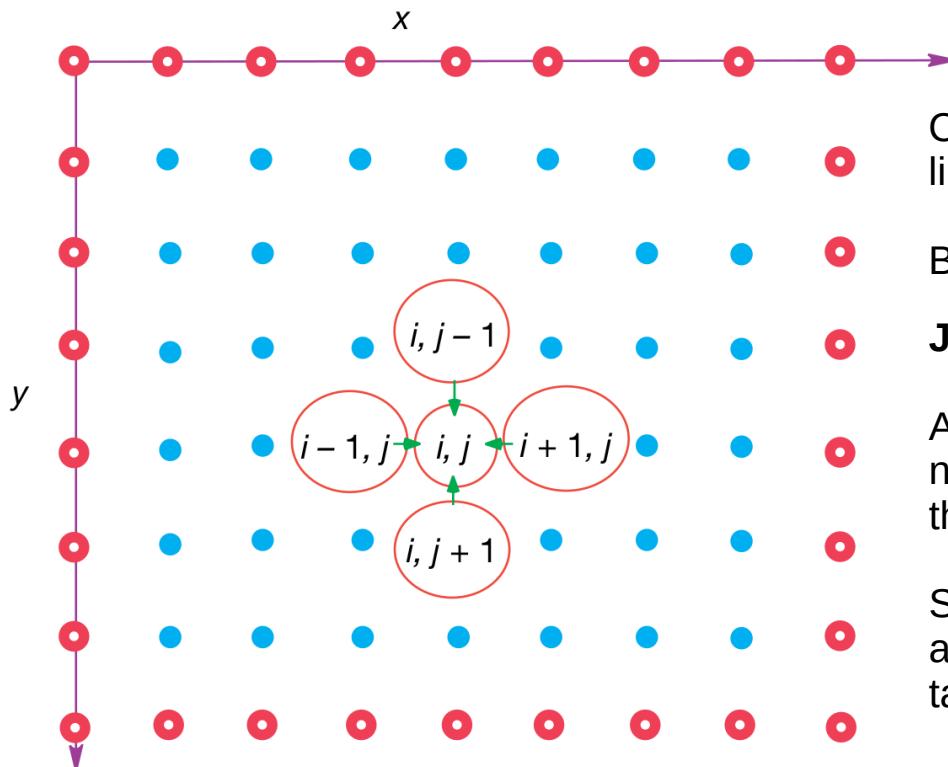
Can solve these equations numerically using linear algebra methods

But, there is a better way (less memory)

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

$$U_{i,j} = \frac{1}{4} [U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}] + \pi\rho(i\Delta, j\Delta)\Delta^2$$



Can solve these equations numerically using linear algebra methods

But, there is a better way (less memory)

Jacobi Update:

A proper solution will be the **average** of the nearest neighbors plus a contribution from the charge density

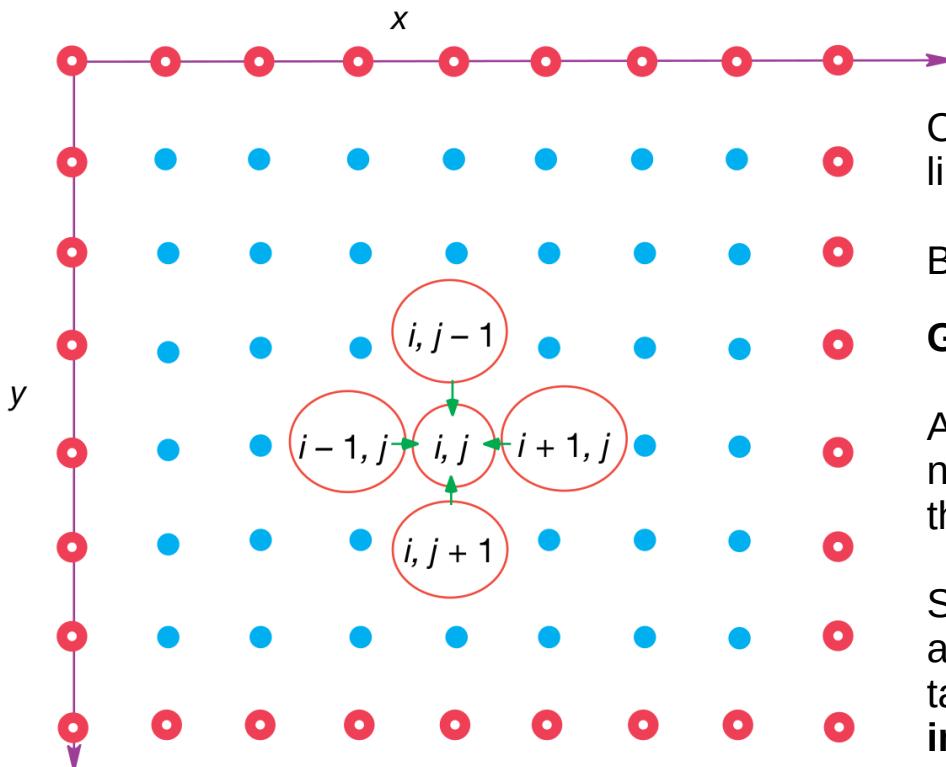
So, start with an initial guess for the potential and improve it by striding through the lattice taking the average over nearest neighbors

Keep doing this until solution converges

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)

$$U_{i,j}^{(\text{new})} = \frac{1}{4} \left[U_{i+1,j}^{(\text{old})} + U_{i-1,j}^{(\text{new})} + U_{i,j+1}^{(\text{old})} + U_{i,j-1}^{(\text{new})} \right]$$



Can solve these equations numerically using linear algebra methods

But, there is a better way (less memory)

Gauss-Seidel Update:

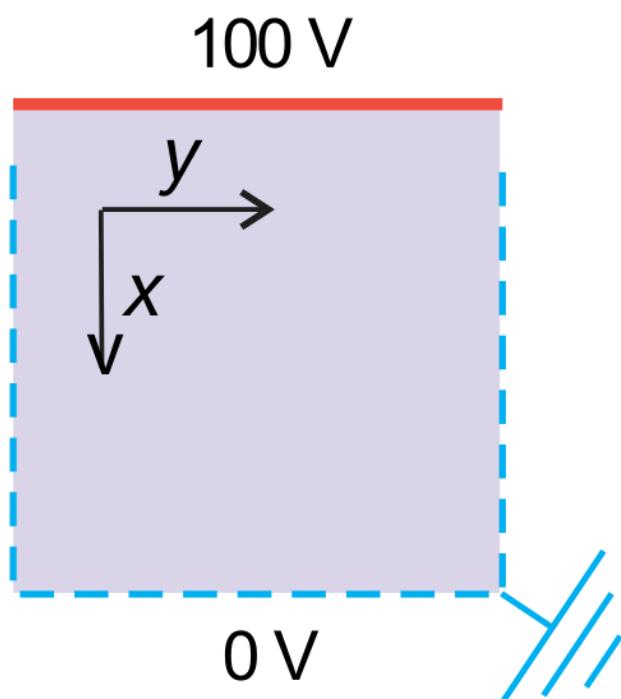
A proper solution will be the **average** of the nearest neighbors plus a contribution from the charge density

So, start with an initial guess for the potential and improve it by striding through the lattice taking the average over nearest neighbors **in real time**.

Keep doing this until solution converges

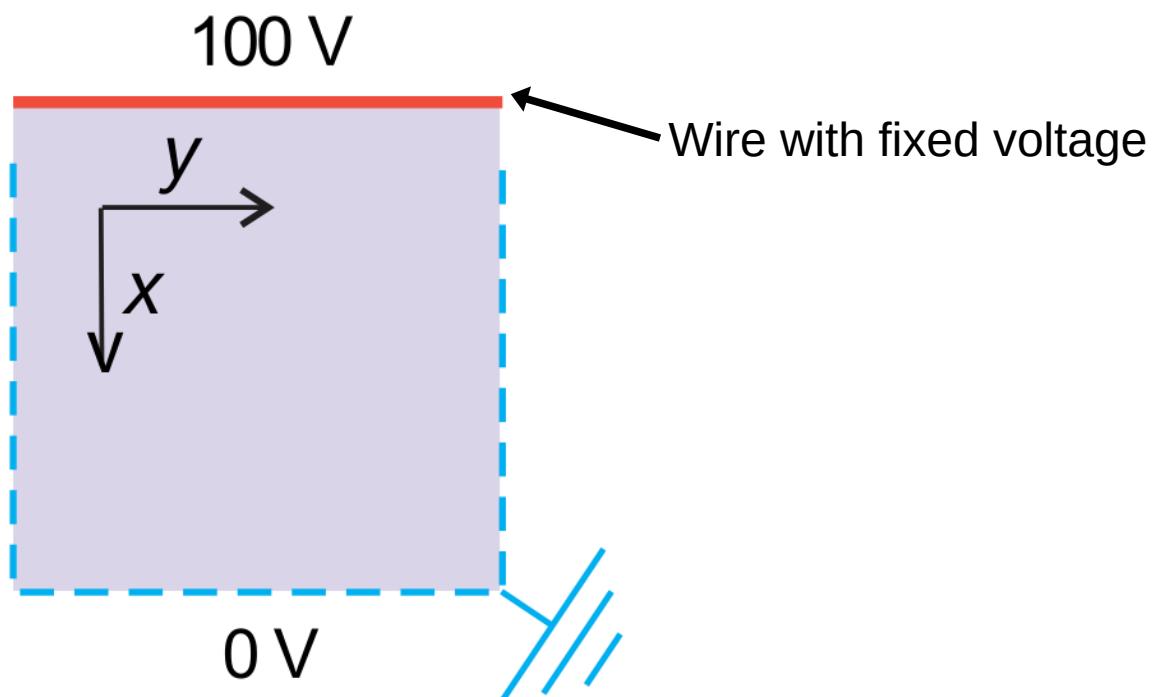
Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



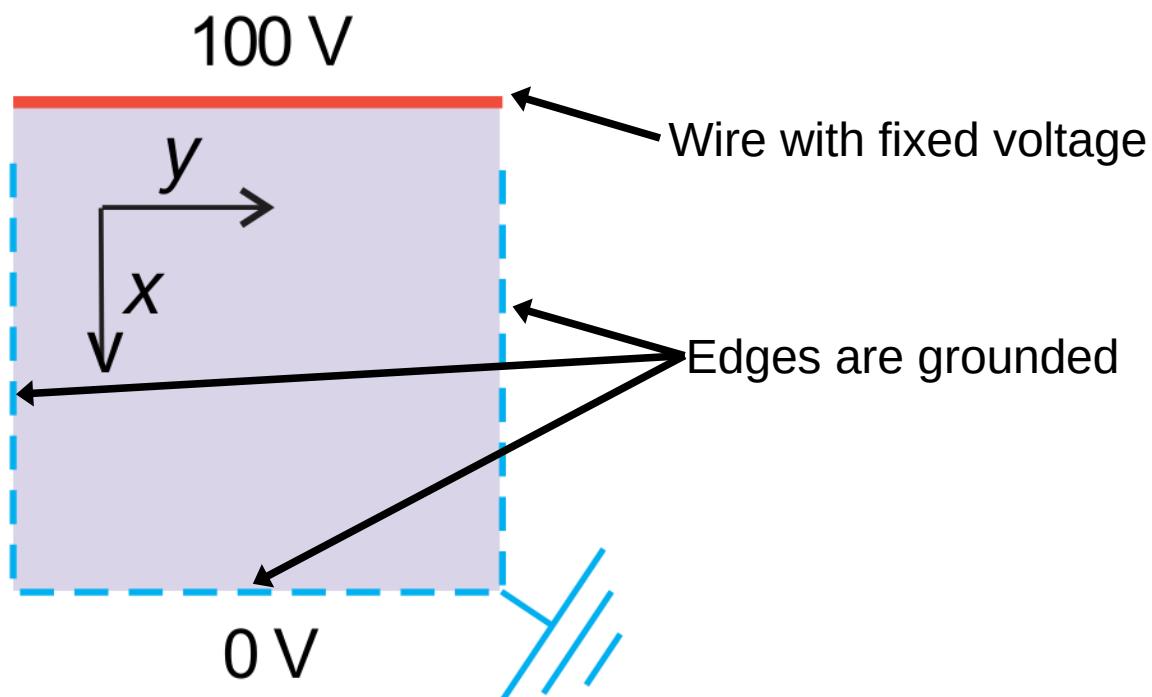
Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



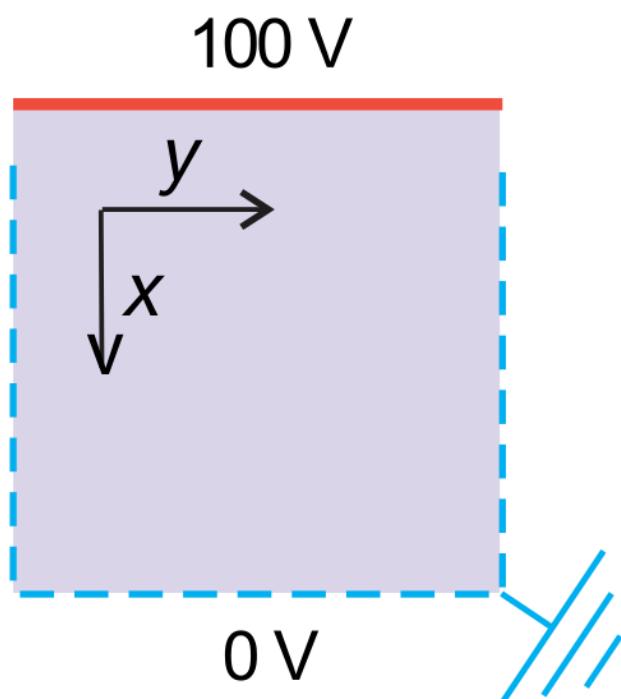
Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



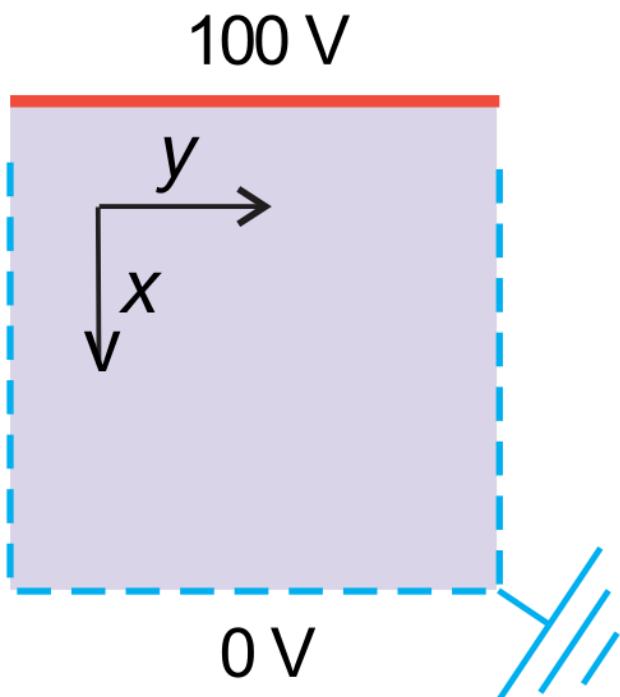
Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12,8)
plt.rcParams['font.size'] = 20

Nmax = 100
V_wire = 100

V = np.zeros((Nmax, Nmax))
V[:,0] = V_wire # Line at V_wire volts

print ("Running the Jacobi Update Solver ...")

def jacobi_update(V, Niter=70):
    Nmax = V.shape[0]
    for iter in range(Niter):
        for i in range(1, Nmax-2):
            for j in range(1,Nmax-2):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])
    return V

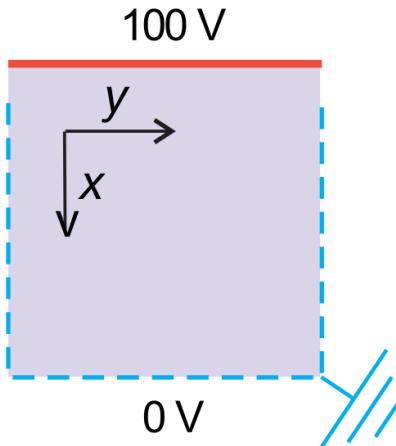
V = jacobi_update(V)
x = np.arange(0, Nmax-1, 2); y = np.arange(0, 50, 2)
X, Y = np.meshgrid(x,y)

def functz(V): # V(x, y)
    z = V[X,Y]
    return z

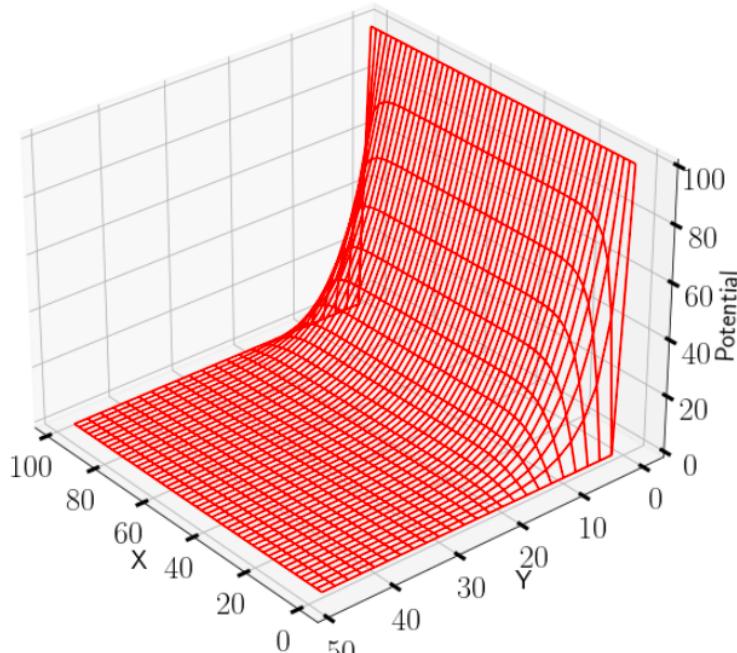
Z = functz(V)
ax = plt.figure().add_subplot(projection='3d')
ax.plot_wireframe(X, Y, Z, color = 'r') # Red wireframe
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Potential')
ax.view_init(30, 140)
plt.show() #Perspective
# Show fig
```

Partial Differential Equations

Finite Difference Methods: 2D Poisson Equation (Dirichlet)



$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$



```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12,8)
plt.rcParams['font.size'] = 20

Nmax = 100
V_wire = 100

V = np.zeros((Nmax, Nmax))
V[:,0] = V_wire # Line at V_wire volts

print ("Running the Jacobi Update Solver ...")

def jacobi_update(V, Niter=70):
    Nmax = V.shape[0]
    for iter in range(Niter):
        for i in range(1, Nmax-2):
            for j in range(1,Nmax-2):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])
    return V

V = jacobi_update(V)
x = np.arange(0, Nmax-1, 2); y = np.arange(0, 50, 2)
X, Y = np.meshgrid(x,y)

def functz(V):
    z = V[X,Y]
    return z # V(x, y)

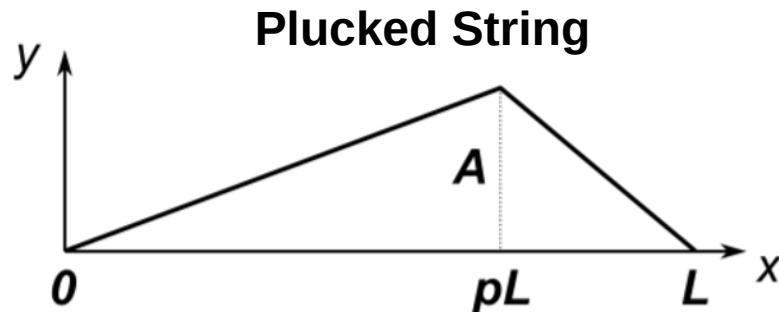
Z = functz(V)
ax = plt.figure().add_subplot(projection='3d')
ax.plot_wireframe(X, Y, Z, color = 'r') # Red wireframe
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Potential')
ax.view_init(30, 140)
plt.show() #Perspective # Show fig
```

See Jupyter notebook: "PDE_FDM.ipynb"

Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$



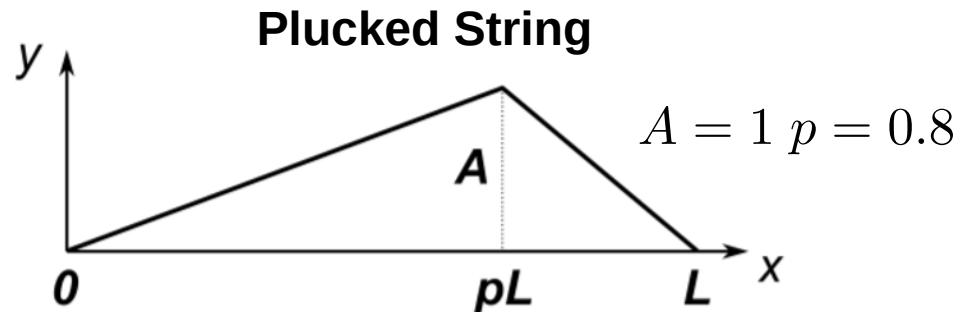
Time-Stepping (Leapfrog Method):

- 1) Divide the space-time domain into a discrete lattice
- 2) Replace the partial derivatives by finite difference approximations evaluated at the lattice points
- 3) Reduce the PDE to a system of linear algebra equations, with initial/boundary conditions added as extra equations
- 4) Jacobi-update one time-step at a time, updating space-steps from the previous time
- 5) Repeat for all times

Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$



Time-Stepping (Leapfrog Method):

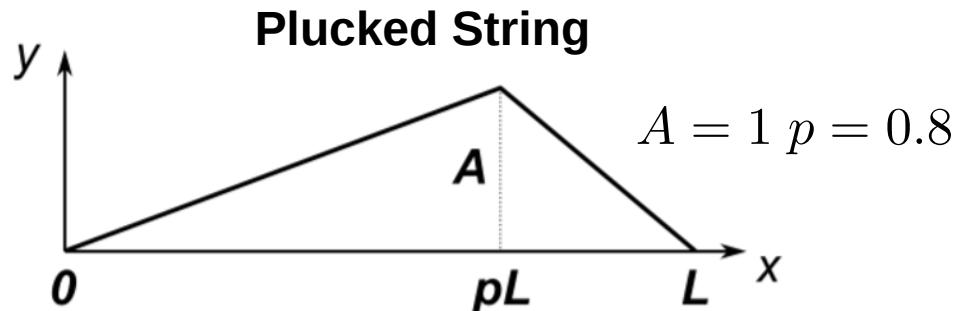
- 1) Divide the space-time domain into a discrete lattice
- 2) Replace the partial derivatives by finite difference approximations evaluated at the lattice points
- 3) Reduce the PDE to a system of linear algebra equations, with initial/boundary conditions added as extra equations
- 4) Jacobi-update one time-step at a time, updating space-steps from the previous time
- 5) Repeat for all times

$$y(x, t = 0) = \begin{cases} 1.25x/L, & x \leq 0.8L, \\ (5 - 5x/L), & x > 0.8L, \end{cases}$$
$$\frac{\partial y}{\partial t}(x, t = 0) = 0$$

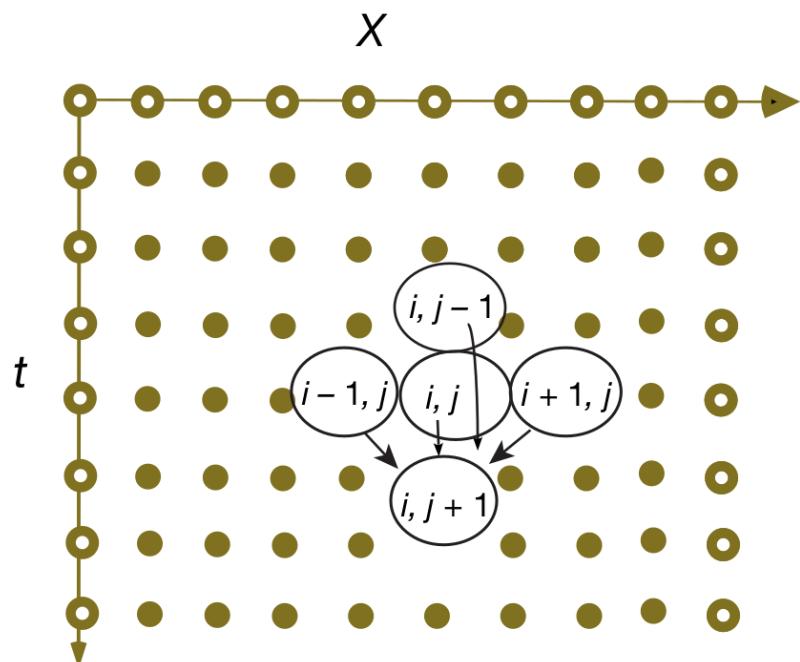
Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$



Time-Stepping (Leapfrog Method):



$$\frac{\partial^2 y}{\partial t^2} \simeq \frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{(\Delta t)^2}, \quad \frac{\partial^2 y}{\partial x^2} \simeq \frac{y_{i+1,j} + y_{i-1,j} - 2y_{i,j}}{(\Delta x)^2}$$

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

Contains three time values:

- $j + 1$ = the future,
- j = the present,
- $j - 1$ = the past

Predict the future solution from the present and past solutions

$$y(x, t = 0) = \begin{cases} 1.25x/L, & x \leq 0.8L, \\ (5 - 5x/L), & x > 0.8L, \end{cases}$$

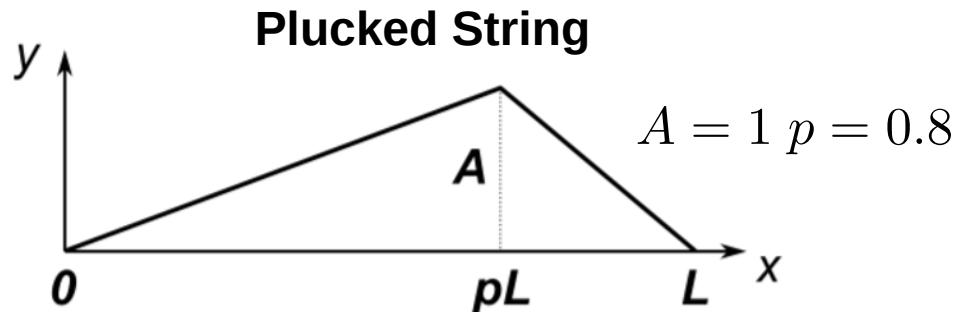
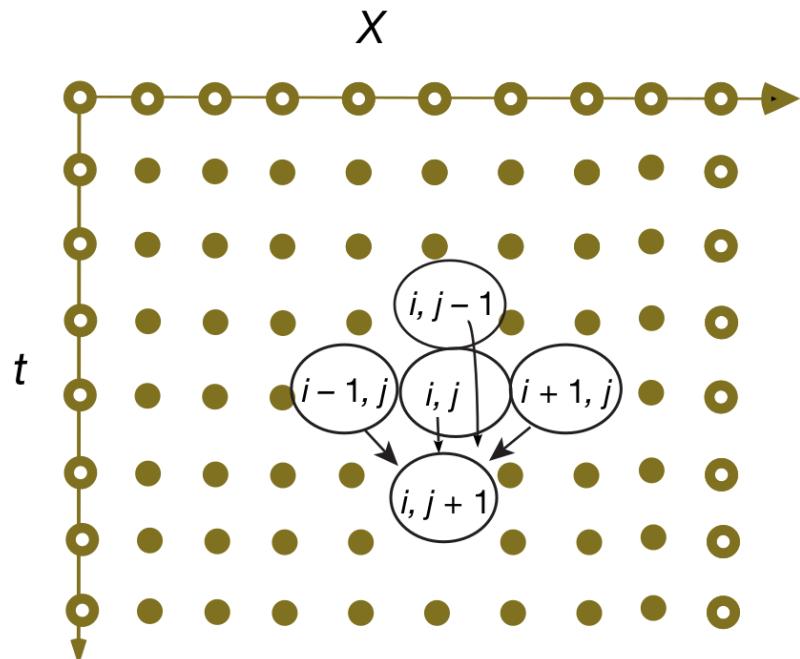
$$\frac{\partial y}{\partial t}(x, t = 0) = 0$$

Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$

Time-Stepping (Leapfrog Method):



$$y(x, t = 0) = \begin{cases} 1.25x/L, & x \leq 0.8L, \\ (5 - 5x/L), & x > 0.8L, \end{cases}$$

$$\frac{\partial y}{\partial t}(x, t = 0) = 0$$

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

↓

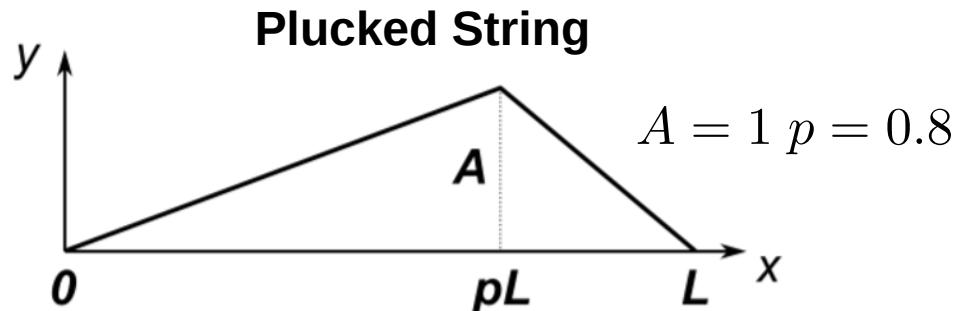
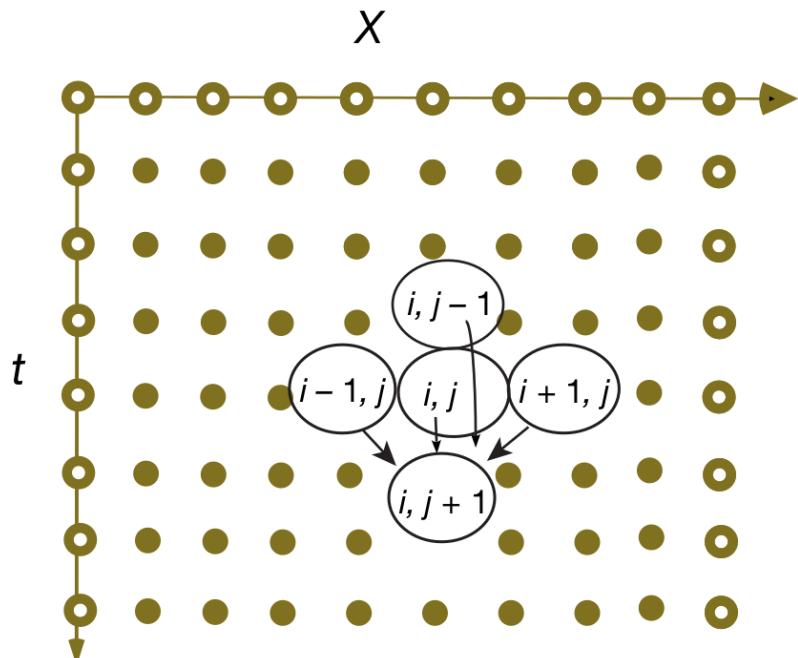
$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \frac{c^2}{c'^2} [y_{i+1,j} + y_{i-1,j} - 2y_{i,j}] , \quad c' \stackrel{\text{def}}{=} \frac{\Delta x}{\Delta t}$$

Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$

Time-Stepping (Leapfrog Method):



$$y(x, t=0) = \begin{cases} 1.25x/L, & x \leq 0.8L, \\ (5 - 5x/L), & x > 0.8L, \end{cases}$$

$$\frac{\partial y}{\partial t}(x, t=0) = 0$$

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \frac{c^2}{c'^2} [y_{i+1,j} + y_{i-1,j} - 2y_{i,j}] , \quad c' \stackrel{\text{def}}{=} \frac{\Delta x}{\Delta t}$$

Von-Neumann Stability Analysis
Ensures that the eigenfuncs don't blow up

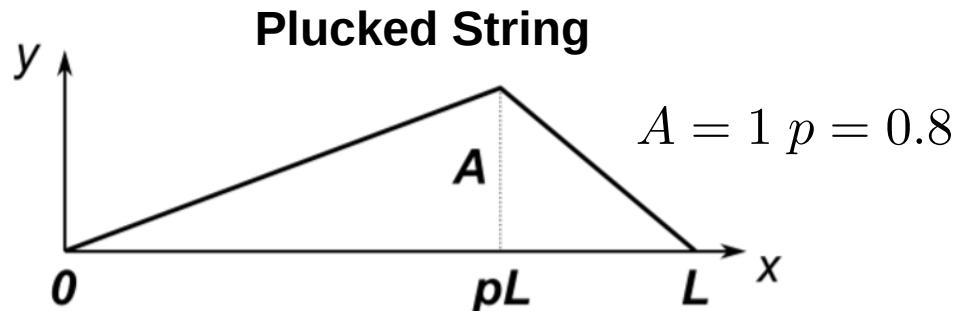
$$c \leq c' = \frac{\Delta x}{\Delta t} \quad \text{Courant condition}$$

Solution gets better with smaller time steps
But gets worse for smaller space steps

Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$



Time-Stepping (Leapfrog Method):

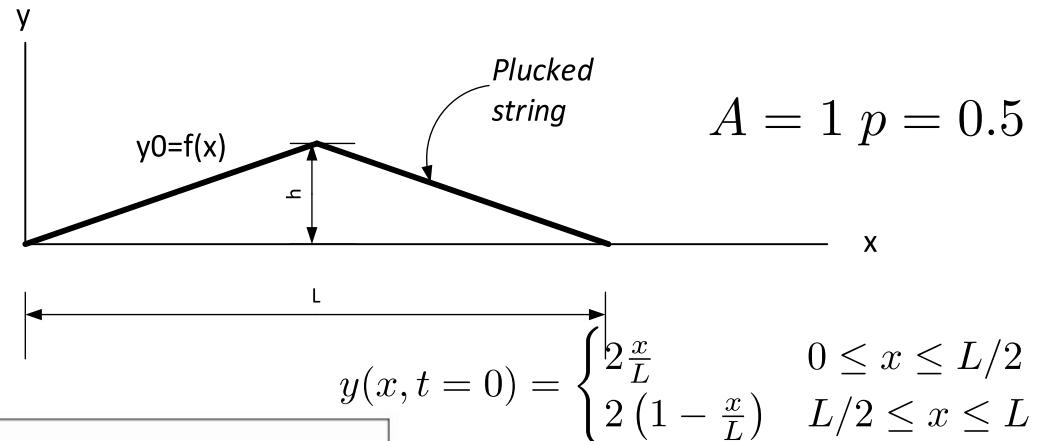
$$y(x, t = 0) = \begin{cases} 1.25x/L, & x \leq 0.8L, \\ (5 - 5x/L), & x > 0.8L, \end{cases}$$

Homework: See if you can change the code to pluck the string in the middle

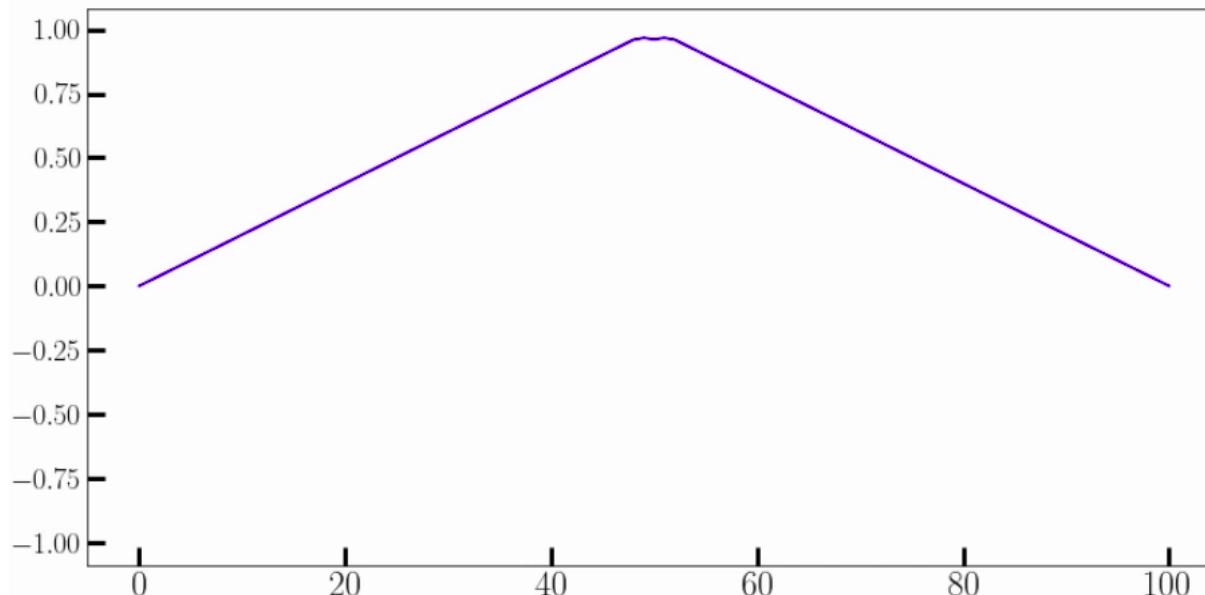
Partial Differential Equations

Finite Difference Methods: 1D Wave Equation (Dirichlet)

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$



Time-Stepping (Leapfrog Method):



Homework: See if you can change the code to pluck the string in the middle

Partial Differential Equations

Finite Element Methods: 2D Poisson Equation (Dirichlet)

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

- 1) Reduce the PDE to an integral equation using Gauss/Stoke's theorem on the boundary
- 2) Divide the domain into a discrete mesh
- 3) Choose basis functions, usually polynomials, corresponding spatially to each mesh element
- 4) Write the RHS as a (known) linear superposition of these basis functions, and the LHS as a superposition with unknown coefficients
- 5) Substitute into the Integral equation and reduce it to a system of linear equations
- 6) Solve using BLAS or whatever.

Partial Differential Equations

Finite Element Methods: 2D Poisson Equation (Dirichlet)

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

- 1) Reduce the PDE to an integral equation using Gauss/Stoke's theorem on the boundary
- 2) Divide the domain into a discrete mesh
- 3) Choose basis functions, usually polynomials, corresponding spatially to each mesh element
- 4) Write the RHS as a (known) linear superposition of these basis functions, and the LHS as a superposition with unknown coefficients
- 5) Substitute into the Integral equation and reduce it to a system of linear equations
- 6) Solve using BLAS or whatever.