The Jacobi Method

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nn}x_n &= b_n \end{aligned}$$

$$\boldsymbol{Ax = b}$$

The matrix A has no zeros on its diagonal: $a_{ii} \neq 0$.

If all off-diagonal elements are 0: $\boldsymbol{x} = A^{-1}\boldsymbol{b}$

If not:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots a_{2n}x_n) \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots a_{n,n-1}x_{n-1}) \end{aligned}$$

$$A = D + R$$

$$\boxed{\boldsymbol{x} = D^{-1}(\boldsymbol{b} - R\boldsymbol{x})}$$

The beauty of numpy arrays (1D, 2D, … *n*D) is their mathematical operations can often be implemented numerically "as written".

Iterative approach

$$\boldsymbol{x}^{(k+1)} \cong D^{-1}(\boldsymbol{b} - R\boldsymbol{x}^{(k)})$$

The initial guess can be nearly arbitrary. In some convention, start with

$$\boldsymbol{x}^{(0)} = D^{-1}\boldsymbol{b} \qquad \text{or, } \boldsymbol{x}^{(0)} = \{0\}$$

Improving Speed: The Gauss-Seidel Method

From *Numerical Recipes* §20.5:

$$A\boldsymbol{x} = \boldsymbol{b} \qquad (20.5.7)$$

It's clear that mathematically Eq 20.5.9 is equivalent to Eq 20.5.13, numerically it's a faster algorithm!

we can consider splitting A as

$$A = L + D + U \quad (20.5.8)$$

where D is the diagonal part of A, L is the lower triangle of A with zeros on the diagonal, and U is the upper triangle of A with zeros on the diagonal.

In the Jacobi method we write for the rth step of iteration

$$\mathbf{D} \cdot \mathbf{x}^{(r)} = -\overbrace{(\mathbf{L} + \mathbf{U})}^{R} \cdot \mathbf{x}^{(r-1)} + \mathbf{b} \qquad (20.5.9)$$

The Gauss-Seidel method, equation (20.5.6), corresponds to the matrix decomposition

$$(\mathbf{L} + \mathbf{D}) \cdot \mathbf{x}^{(r)} = -\mathbf{U} \cdot \mathbf{x}^{(r-1)} + \mathbf{b} \qquad (20.5.13)$$

The fact that $\mathbf{L}$ is on the left-hand side of the equation follows from the updating in place, as you can easily check if you write out (20.5.13) in components.

What's powerful about linear algebra in general, and the Jacobi Method in particular:

There is little difference between solving a set of linear algebraic equations and solving *differential* equations!

Numerical Second Derivatives and the Laplacian

$$f'' = \frac{\dfrac{f(x+h)-f(x)}{h} - \dfrac{f(x)-f(x-h)}{h}}{h} = \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

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

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} \qquad \left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2}$$

Turning the Laplace Equation into a Linear Algebra Problem

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

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2} = 0$$

Setting $\Delta x = \Delta y$,

$$-4u_{i,j} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} = 0$$

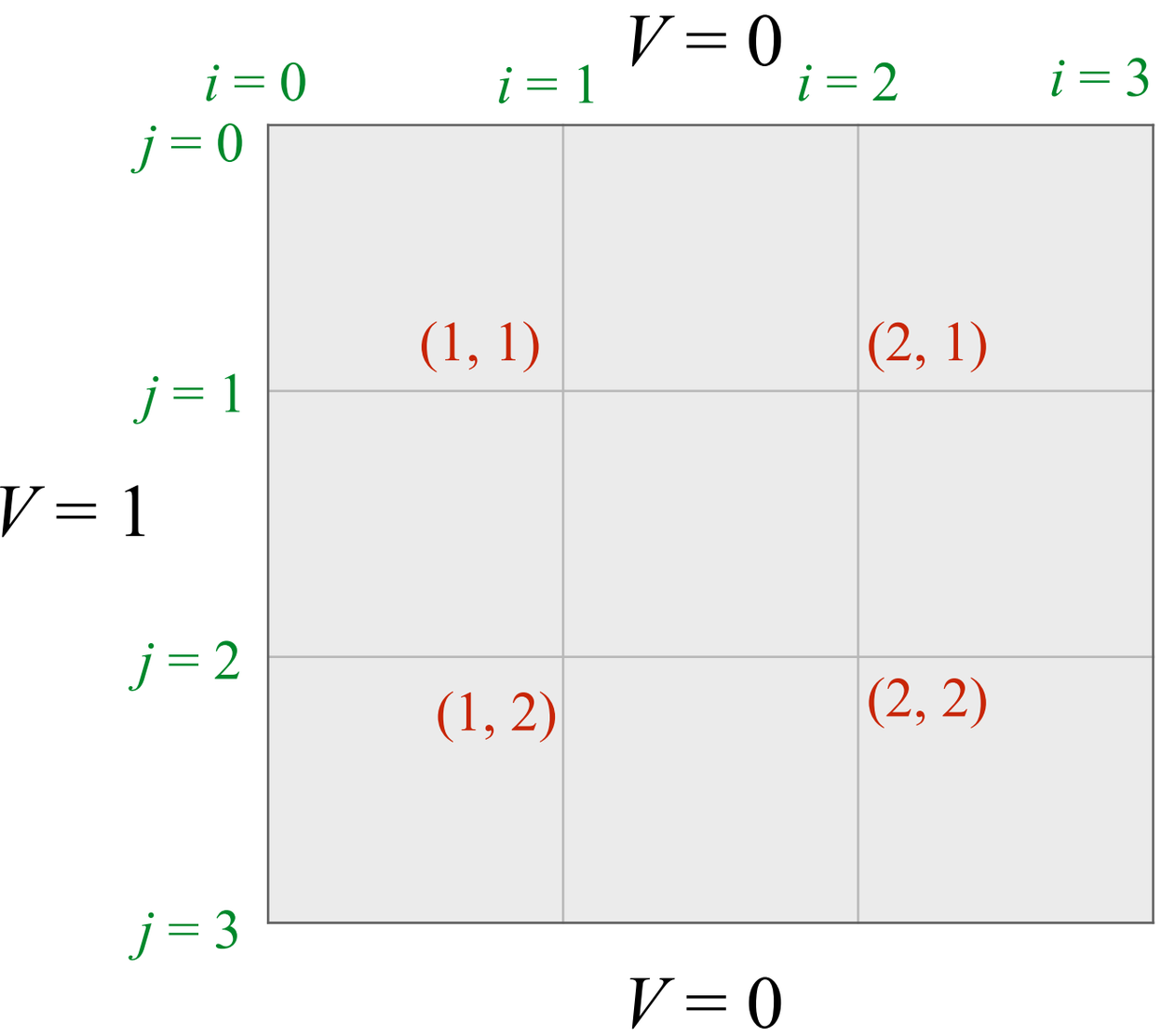
Note: This is saying,

$$u_{i,j} = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})/4$$

Solving the Laplace Equation the Matrix Way

$$-4\,u_{i,j} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} = 0$$

Note: u_{ij} is the average of its neighboring points



$$\begin{aligned}
-4\,u_{1,1} + u_{1,2} + u_{2,1} + u_{0,1} + u_{1,0} &= 0 \\
u_{1,1} - 4\,u_{1,2} + u_{2,2} + u_{0,2} + u_{1,3} &= 0 \\
u_{1,2} - 4\,u_{2,2} + u_{2,1} + u_{2,3} + u_{3,2} &= 0 \\
u_{1,1} + u_{2,2} - 4\,u_{2,1} + u_{2,0} + u_{3,1} &= 0
\end{aligned}$$

$$\begin{pmatrix} -4 & 1 & 0 & 1 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,2} \\ u_{2,1} \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

A sparse matrix!

It has the form: $\mathbf{A} \cdot \mathbf{u} = \mathbf{b}$

Jacobi Method!
Or Gauss-Seidel Method!

Generalize to $(J+1)\times(L+1)$ Grid

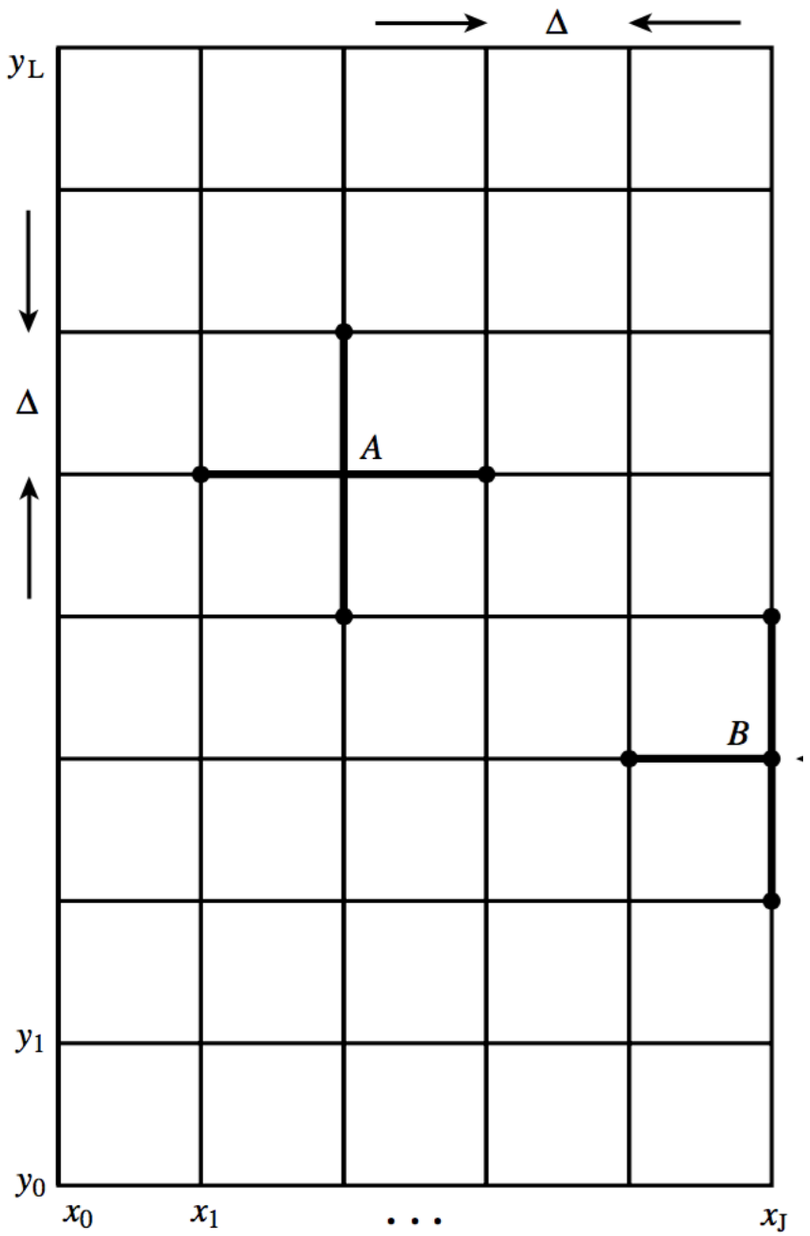
That is,

$$x_j = x_0 + j\Delta, \qquad j = 0, 1, \ldots, J$$
$$y_l = y_0 + l\Delta, \qquad l = 0, 1, \ldots, L$$

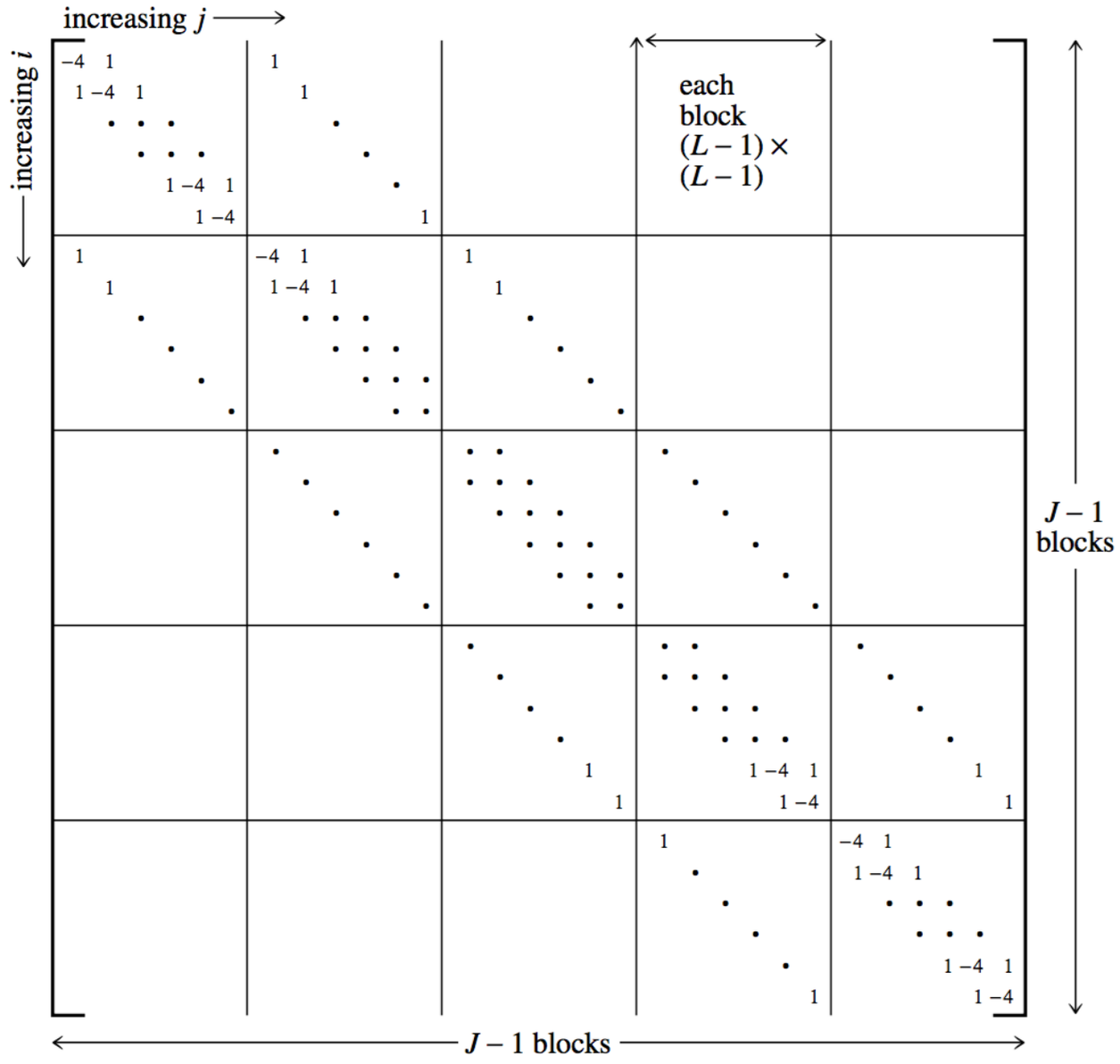
There are $(J-1)\times(L-1)$ interior points, or $(J-1)\times(L-1)$ unknowns

→ expect a $[(J-1)\times(L-1)] \times [(J-1)\times(L-1)]$ matrix

The Problem



The Matrix



Sparse!

(NR 3rd ed., p. 1028- 1029)

Numerical Resolution → Size of Matrix

E.g., 1000 by 1000 grid will give you linear fractional resolution of 0.001

This requires a 1 million by 1 million matrix!

→ you don't want to simply use brute-force matrix inversion, but should use faster algorithms like Gauss-Seidel.

What's More…

Generally 2nd PDE can be written as

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0$$

Note: F is the inhomogeneity.

Every single such equation can be solved by the Gauss-Seidel Method (in its matrix form), or even more efficient methods

There are two other kinds of PDE's:

- Elliptic ($B - AC < 0$, e.g., Laplace Equation)
- Hyperbolic ($B - AC > 0$, e.g., Wave Equation)
- Parabolic ($B - AC = 0$, e.g., Diffusion Equation)

We will talk about how to solve the other two kinds in CP-II…

Breakout Problem

Part I: Solve for the u_{ij}'s for the interior points using the Jacobi Method

$$\begin{pmatrix} -4 & 1 & 0 & 1 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,2} \\ u_{2,1} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

An equivalent approach for the Laplace Eq (only): u_{ij} is the average of its neighboring points — the "averaging method."

The linear algebra approach is completely equivalent to the "averaging method." If for the averaging method, as you go through each point, the value at each point is updated using *only the neighboring values from the previous iteration*: this is equivalent to the Jacobi Method.

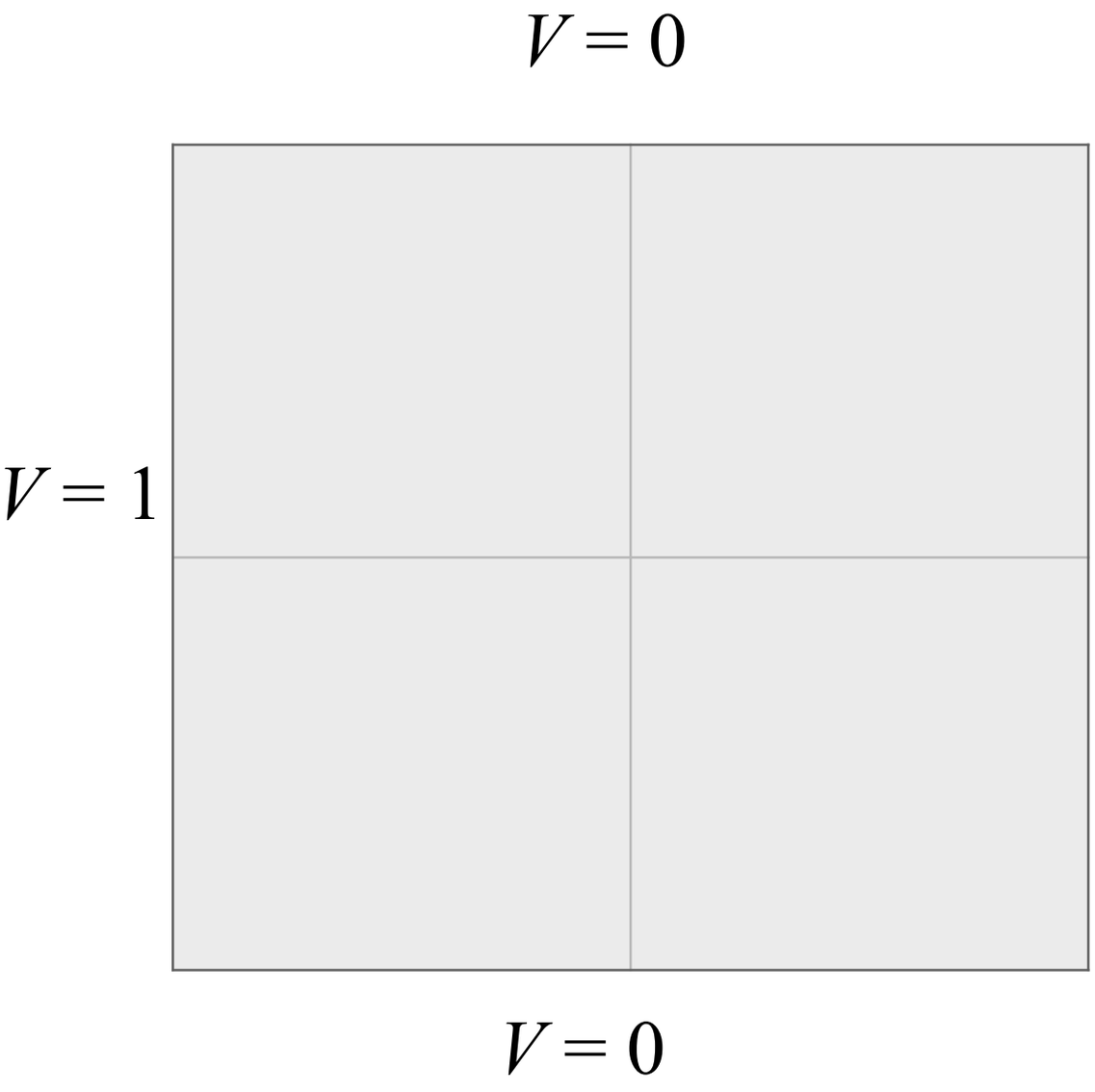
If you use the values for the neighboring points that have just been updated in *this* iteration — this is equivalent to the Gauss-Seidel Method.

Part II: Implement the "averaging method" the Gauss-Seidel way for a 11×11 point grid.

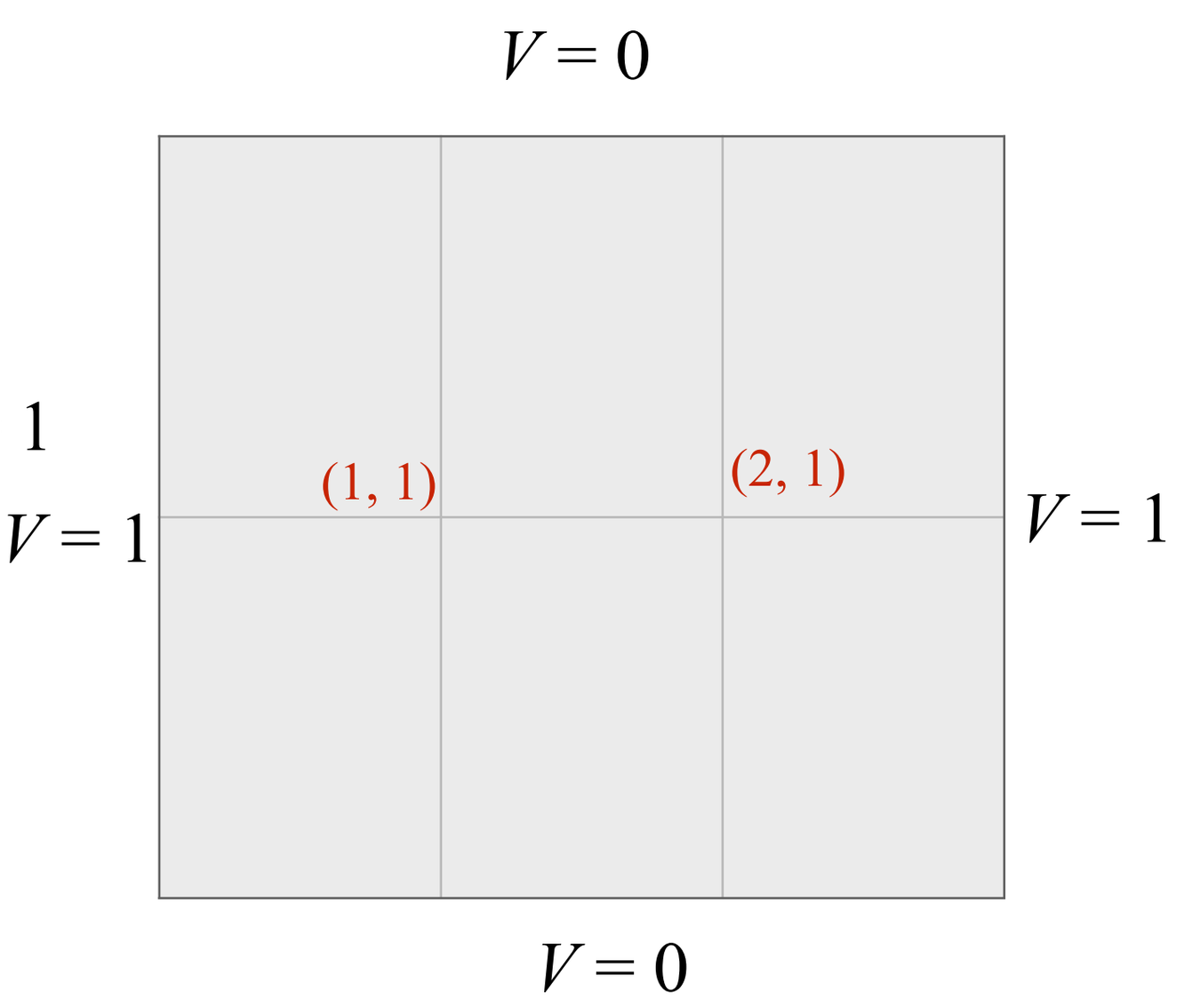
Numerical vs. Analytical Solutions

	Analytical	Numerical
Laplace Equation with simple geometry and BC	✓	✓
Arbitrary Charge Distribution (i.e. Poisson Equation)	Basically impossible	✓
Complicated Geometry	Basically impossible	✓
Complicated BC	Basically impossible	✓

Can You Go Lower Than a 4 by 4 Grid?



An algebraic Problem



A 2 by 2 matrix problem

End of Week 6