Finite Difference Methods

Finite differences in 1D

Sparse matrix formulation

Boundary conditions

Solving two-point boundary value problems

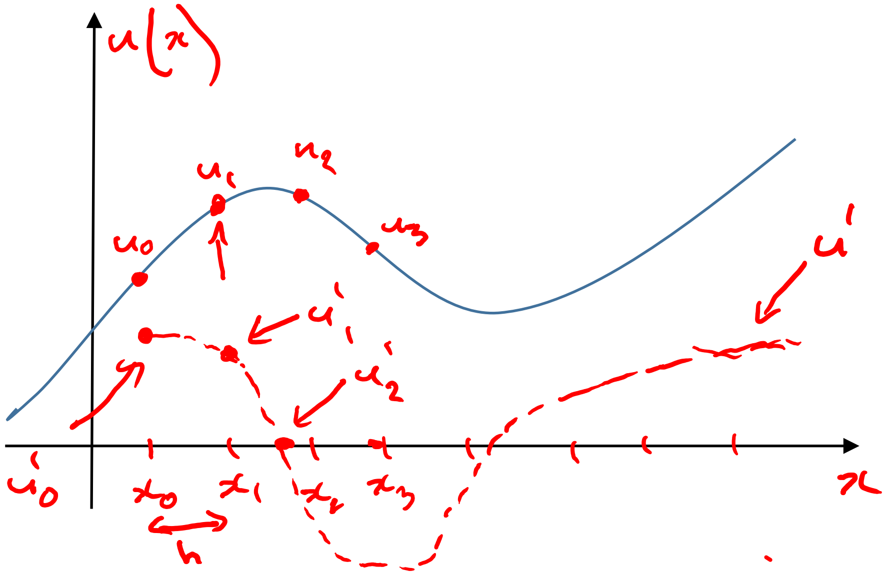
Setting up higher dimensional problems

Finite differences in higher dimensions

Boundary conditions

Other approaches

Imagine we have a function $u(x)$ with known values at a fixed number of equally-spaced nodes, with spacing h:



$$u'_1 = \frac{u_2 - u_0}{2h}$$

$$\frac{du}{dx} = f(x,u)$$

We saw in Week 2 that an approximation for the first derivative of u at the j^{th} node is

$$u'_j = \frac{1}{2h}\left(u_{j+1} - u_{j-1}\right)$$

If we represent u as a *vector*, then the numerical derivative can be represented as a matrix:

$$\underset{\sim}{u'} = \begin{pmatrix} u_0' \\ u_1' \\ u_2' \\ \vdots \\ u_j' \\ \vdots \end{pmatrix} = \begin{pmatrix} -\frac{1}{2h} & 0 & \frac{1}{2h} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & -\frac{1}{2h} & 0 & \frac{1}{2h} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & \ddots & & \ddots & & & & & & \\ \cdots & & & -\frac{1}{2h} & 0 & \frac{1}{2h} & 0 & 0 & 0 & 0 & 0 \\ & & & & & & \ddots & & & & \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_j \\ \vdots \end{pmatrix} \leftarrow \underset{\sim}{u}$$

$$u_j' = \frac{1}{2h}\left(u_{j+1} - u_{j-1}\right)$$

$$\underline{u_2' = \frac{1}{2h}\left(u_3 - u_1\right)}.$$

This is a *tridiagonal, sparse matrix.*

The first-order central difference derivative is

$$u_j' = \frac{1}{2h}\left(-u_{j-1} + u_{j+1}\right)$$

Which leads to the matrix equation

forward difference

$$u_0' = \frac{1}{h}(u_1 - u_0)$$

$$= -\frac{1}{2h}u_0 + \frac{1}{2h}u_1$$

$$\begin{pmatrix} u_0' \\ \vdots \\ \vdots \\ u'_{j-1} \\ u_j' \\ u_{j+1}' \\ \vdots \end{pmatrix} = \frac{1}{2h} \begin{pmatrix} -2 & 2 & & & & & & & \\ -1 & 0 & 1 & & & & & & \\ & & \ddots & & & 0 & & & \\ & & -1 & 0 & 1 & & & & \\ & & & -1 & 0 & 1 & & & \\ & 0 & & & -1 & 0 & 1 & & \\ & & & & & & \ddots & & \\ & & & & & & -1 & 0 & 1 \\ & & & & & & & -2 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ \vdots \\ u_{j-1} \\ u_j \\ u_j{+}1 \\ \vdots \\ u_{N-1} \end{pmatrix}$$

The second-order derivative (using central differences) is

$$u_j'' = \frac{1}{h^2}\left(u_{j-1} - 2\,u_j + u_{j+1}\right)$$

Which leads to the matrix equation

$$\underbrace{\begin{pmatrix} \\ \vdots \\ \vdots \\ u''_{j-1} \\ u_j'' \\ u_{j+1}'' \\ \\ \vdots \\ \\ \end{pmatrix}}_{u''} = \frac{1}{h^2} \overbrace{\begin{pmatrix} 1 & -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & & \\ & \ddots & \ddots & \ddots & & & 0 & \\ & & 1 & -2 & 1 & & & \\ & & & 1 & -2 & 1 & & \\ & 0 & & & 1 & -2 & 1 & \\ & & & & & \ddots & \ddots & \ddots \\ & & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 & 1 \end{pmatrix}}^{D_2} \overbrace{\begin{pmatrix} u_0 \\ \\ \vdots \\ \\ u_{j-1} \\ u_j \\ u_j+1 \\ \vdots \\ \\ u_{N-1} \end{pmatrix}}^{u}$$

We say that D_2 is a <u>finite difference representation</u> of the differential operator

$$L = \frac{d^2}{dx^2}$$

$$u'' = D_2 u$$

Finite differences for boundary value problems

We can solve differential equations numerically by substituting the matrix representations of the operator and then solving as if it were a matrix equation.

E.g. to solve

$$-\frac{d^2u}{dx^2} = f(x)$$

We would set up the matrix equation

$$D_2\mathrm{u} = \mathrm{f}$$

where

$$D_2 = \frac{1}{h^2}\begin{pmatrix} 1 & -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & & \ddots & & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & & \ddots & & \\ & & & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 & 1 \end{pmatrix} \qquad \mathrm{u} = \begin{pmatrix} u_0 \\ \vdots \\ u_{j-1} \\ u_j \\ u_{j+1} \\ \vdots \\ u_{N-1} \end{pmatrix} \qquad \mathrm{f} = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \\ \vdots \\ f(x_{N-1}) \end{pmatrix}$$

To solve this system (i.e. to find the unknowns u_j) we *invert* this equation:

$$u = D_2^{-1} f$$

<u>However</u> we still have to apply <u>Boundary conditions</u>

<u>Boundary Conditions</u> usually take the form

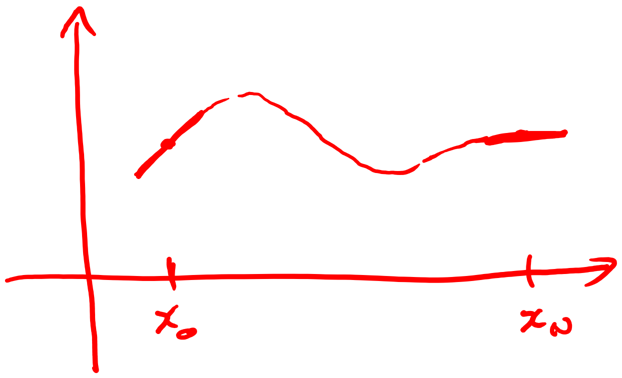
$u(x_0) = A, \quad u(x_{N-1}) = B$ (two-point boundary value problems)

To enforce a two-point BVP we modify the first and last lines of the matrix equation:

$$
\frac{1}{h^2}
\begin{pmatrix}
1 & 0 & 0 & & & & & & \\
1 & -2 & 1 & & & & & & \\
 & & \ddots & & & & & & \\
 & & 1 & -2 & 1 & & & & \\
 & & & 1 & -2 & 1 & & & \\
 & & & & 1 & -2 & 1 & & \\
 & & & & & & \ddots & & \\
 & & & & & & 1 & -2 & 1 \\
 & & & & & & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
u_0 \\ \vdots \\ u_{j-1} \\ u_j \\ u_j+1 \\ \vdots \\ u_{N-1}
\end{pmatrix}
=
\begin{pmatrix}
A \\ \vdots \\ f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \\ \vdots \\ B
\end{pmatrix}
$$

To create a boundary condition with a derivative, e.g.

$$u'(x_0) = C \longrightarrow u'(x_0) = \frac{1}{h}(u_1 - u_0) = \boxed{\frac{1}{h^2}(hu_1) - \frac{1}{h^2}(hu_0) = C}$$



You modify the first (or last) line to approximate the derivative using forward (or backward) differences:

$$\underbrace{\frac{1}{h^2}\begin{pmatrix} -h & +h & 0 & 0 & 0 & 0 & 0 & & & \\ 1 & -2 & 1 & & & & & & & \\ & & \ddots & & & & & & & \\ & & 1 & -2 & 1 & & & & & \\ & & & 1 & -2 & 1 & & & & \\ & & & & 1 & -2 & 1 & & & \\ & & & & & & \ddots & & & \\ & & & & & & & 1 & -2 & 1 \\ & & & & & & & & & \end{pmatrix}}_{D_2^B} \underbrace{\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{j-1} \\ u_j \\ u_j+1 \\ \vdots \\ u_{N-1} \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} C \\ \vdots \\ f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \\ \vdots \\ \ \end{pmatrix}}_{f^B}$$

$$D_2^B u = f^B$$

$$\hookrightarrow \underline{u = (D_2^B)^{-1} f^B}$$

Finite differences can be used to solve almost anything in 1D.

Advantages:

- Simple to implement
- Fast

Disadvantages:

- Requires knowledge of sparse systems
- A bit fiddly with boundary conditions
- Not so good for “stiff” problems (i.e. ones for which the function varies rapidly).

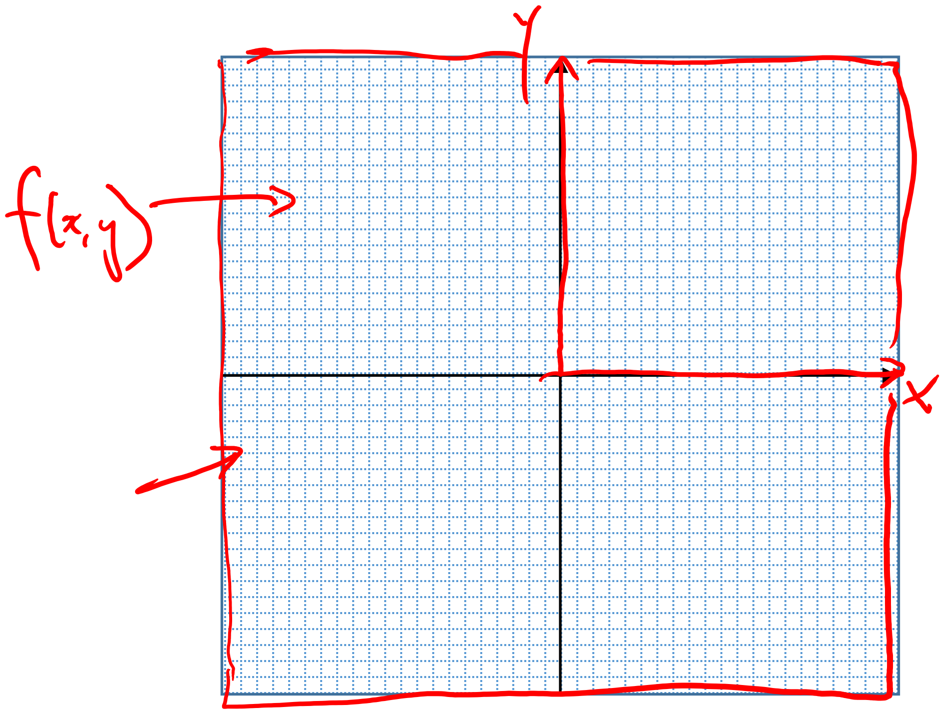
Finite Differences in higher dimensions

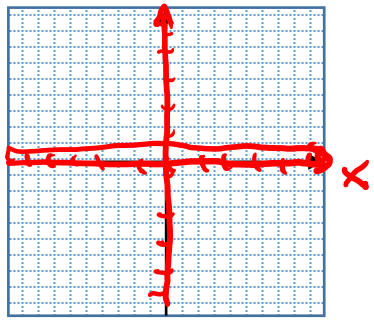
A big strength of the FD method is its ability to solve Partial Differential Equations (PDEs) in 2D and higher dimension.

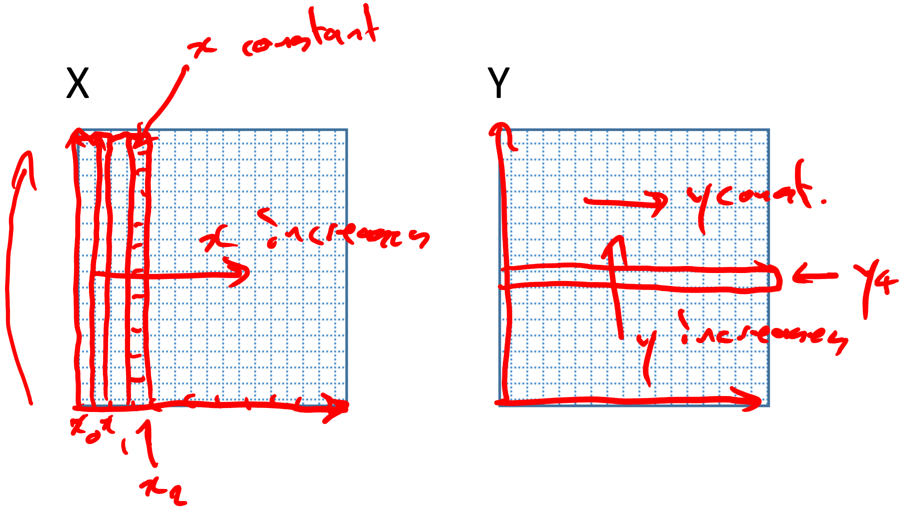
To formulate a FD method in 2D we need to convert a 2D grid into a vector, and back again.

A 2D numpy array can be created using a combination of linspace and meshgrid (See Lab 4):

```
x = np.linspace(-5,5,10)
y = np.linspace(-5,5,10)

X,Y = np.meshgrid(x,y)
```





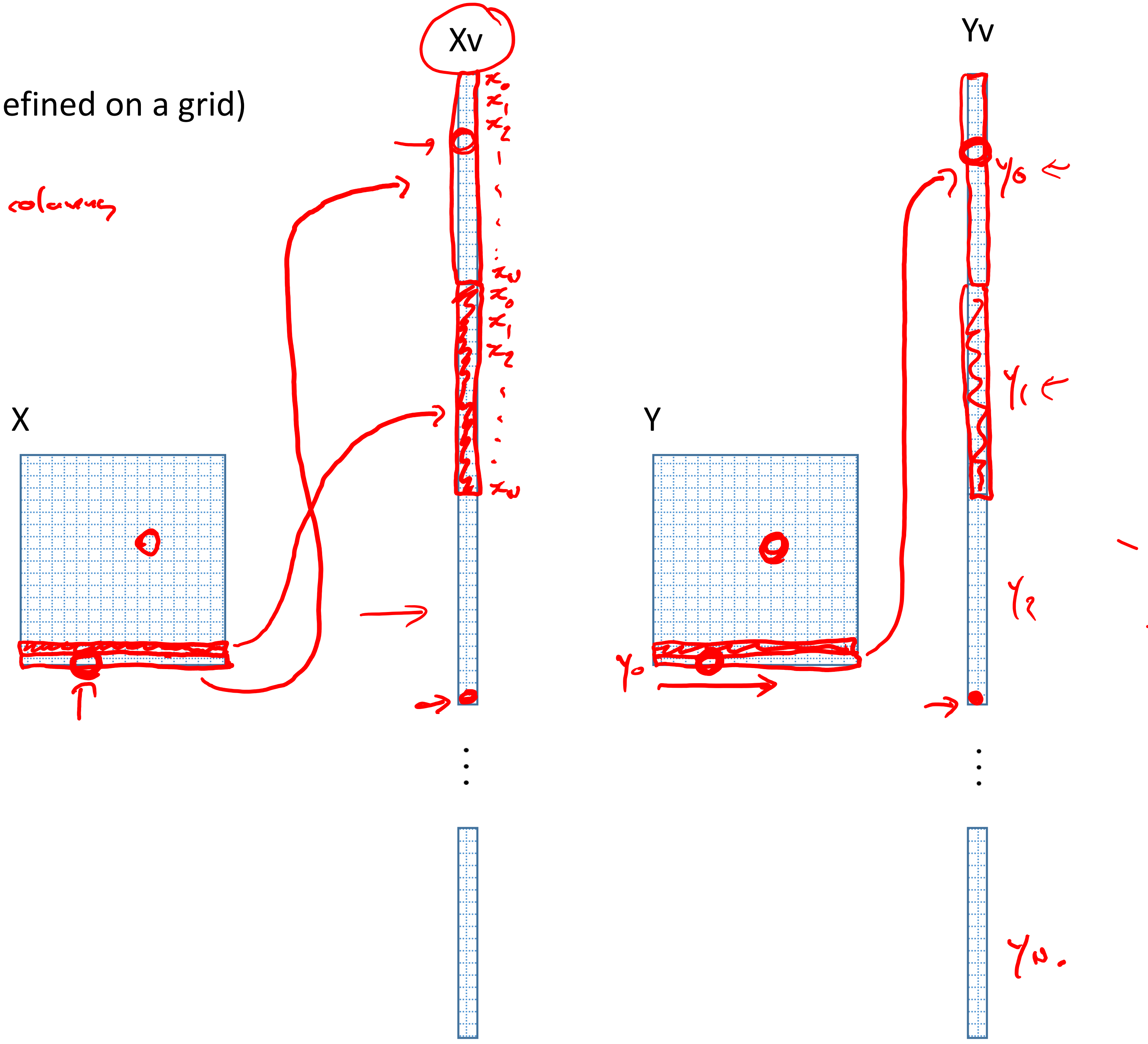


We can then unfold the grid (and any functions defined on a grid) using the numpy *reshape* function:

no. of rows
no. of columns

```
Xv = np.reshape(X,(10*10,1))
Yv = np.reshape(Y,(10*10,1))
```

Create a 100 x 1 column vector



(Whenever you do this, it helps (a lot) to draw this out on paper)

If we have a function defined on a grid this can also be disassembled:

```
def f(x,y):

    f = np.sin(x)*np.cos(y)

    return f
```

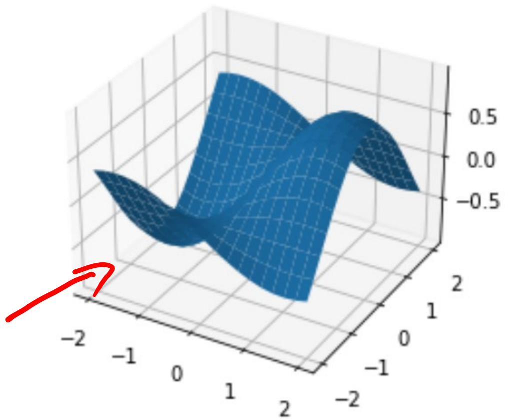
⋮

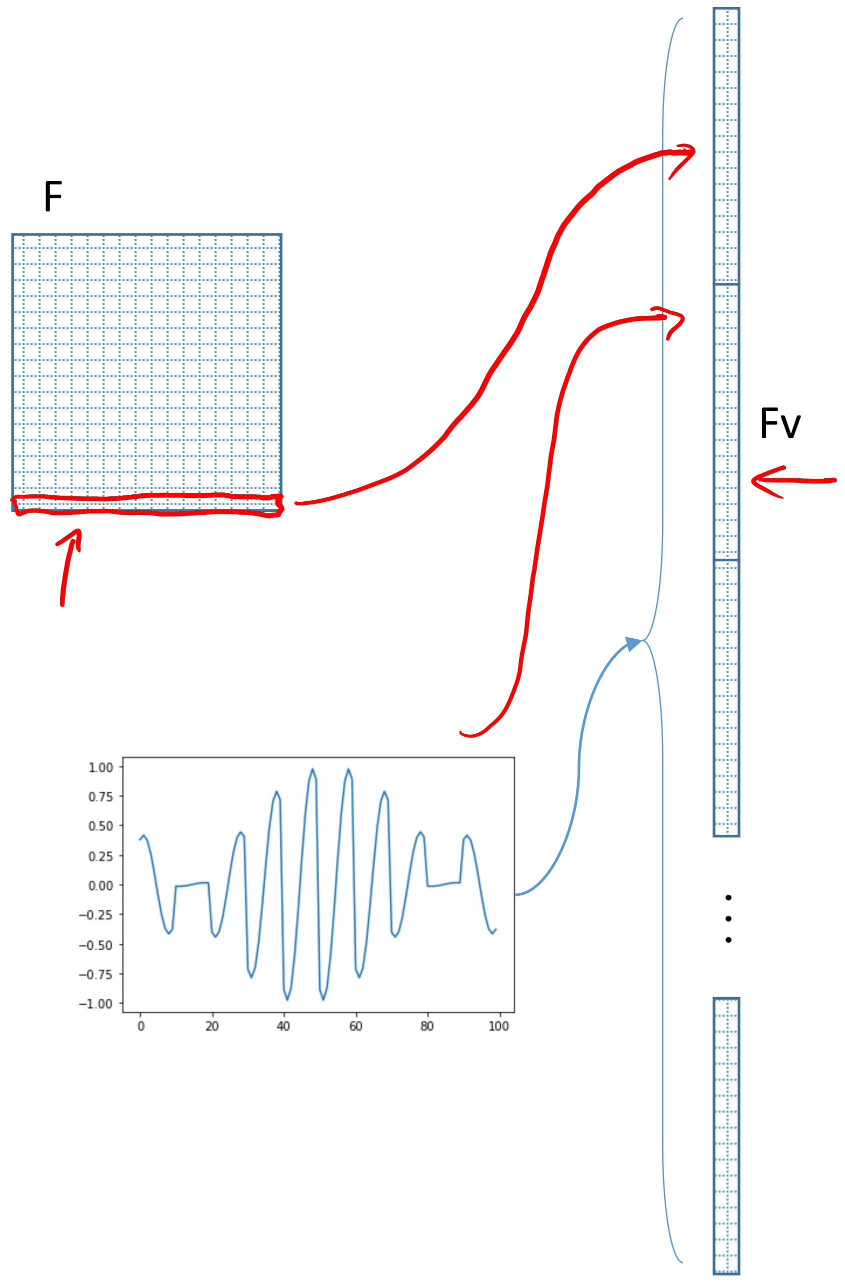
```

F = f(X,Y)

```

2D grid

⋮

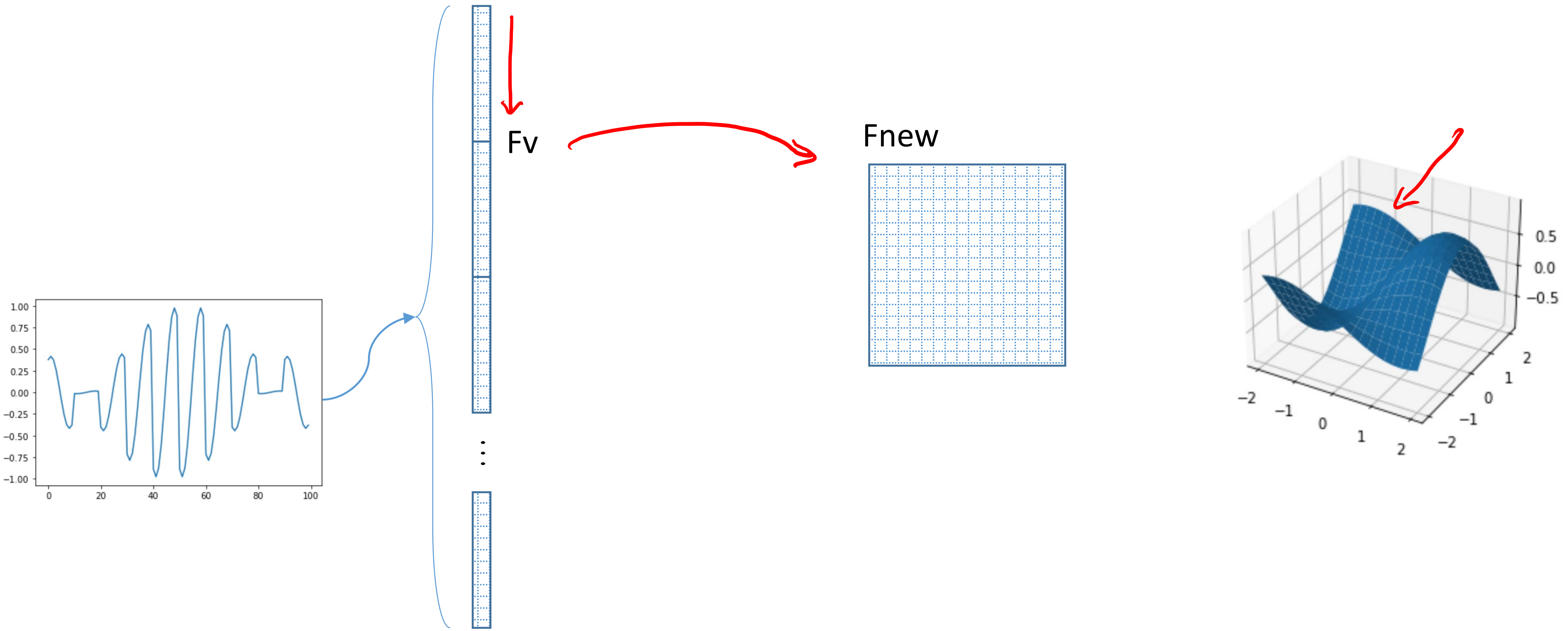
```

Fv = np.reshape(F,(N*N,1))

```





We can then re-assemble the functions into the original grid using reshape:

```
Fnew = np.reshape(Fv,(N,N))
```

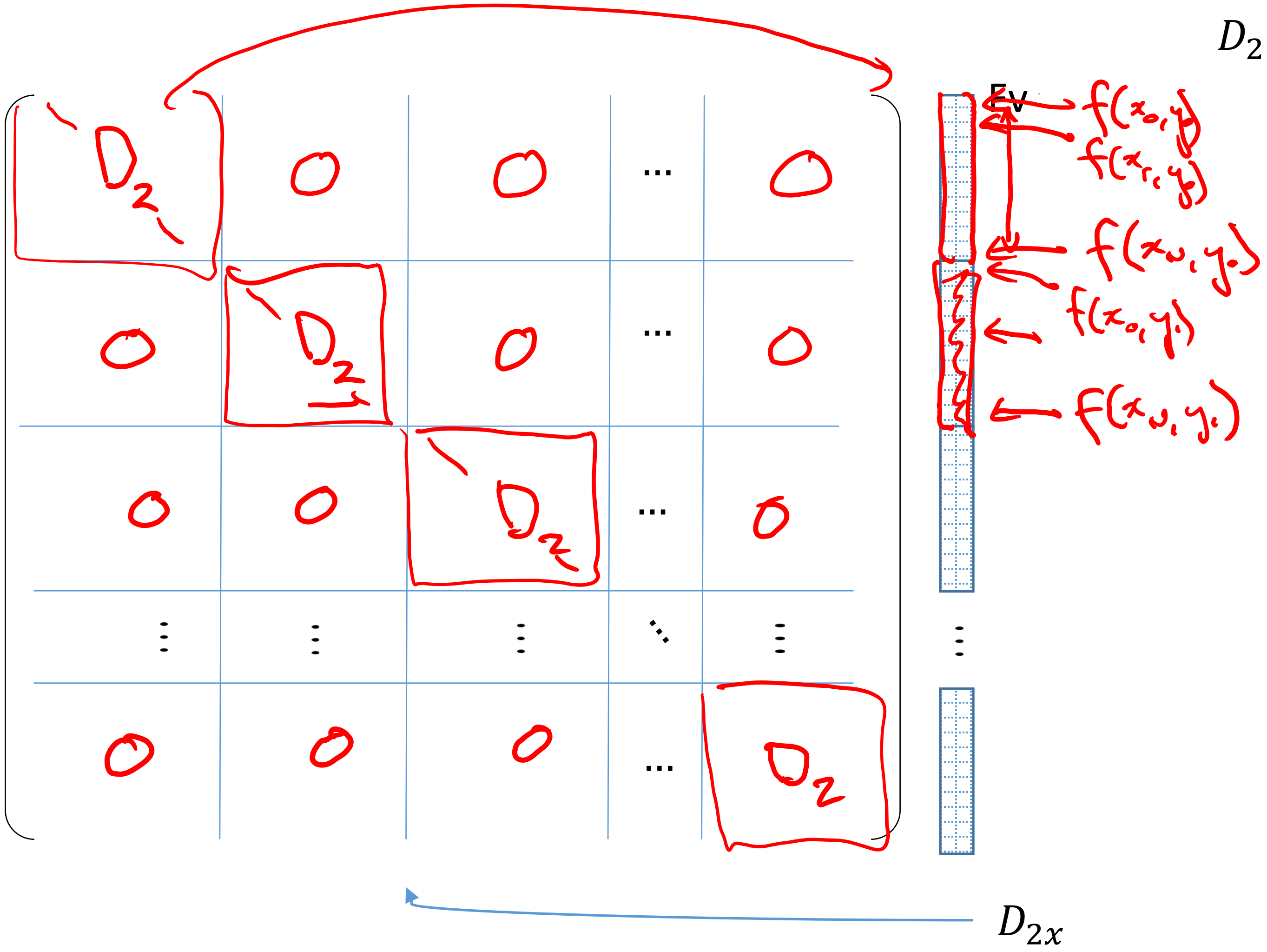


The partial derivative with respect to x

Recall that D_2 gives the 2[nd] partial derivative with respect to x of a single vector $u(x_j)$

To get the partial derivative, we build a block matrix consisting of the D_2 matrices:

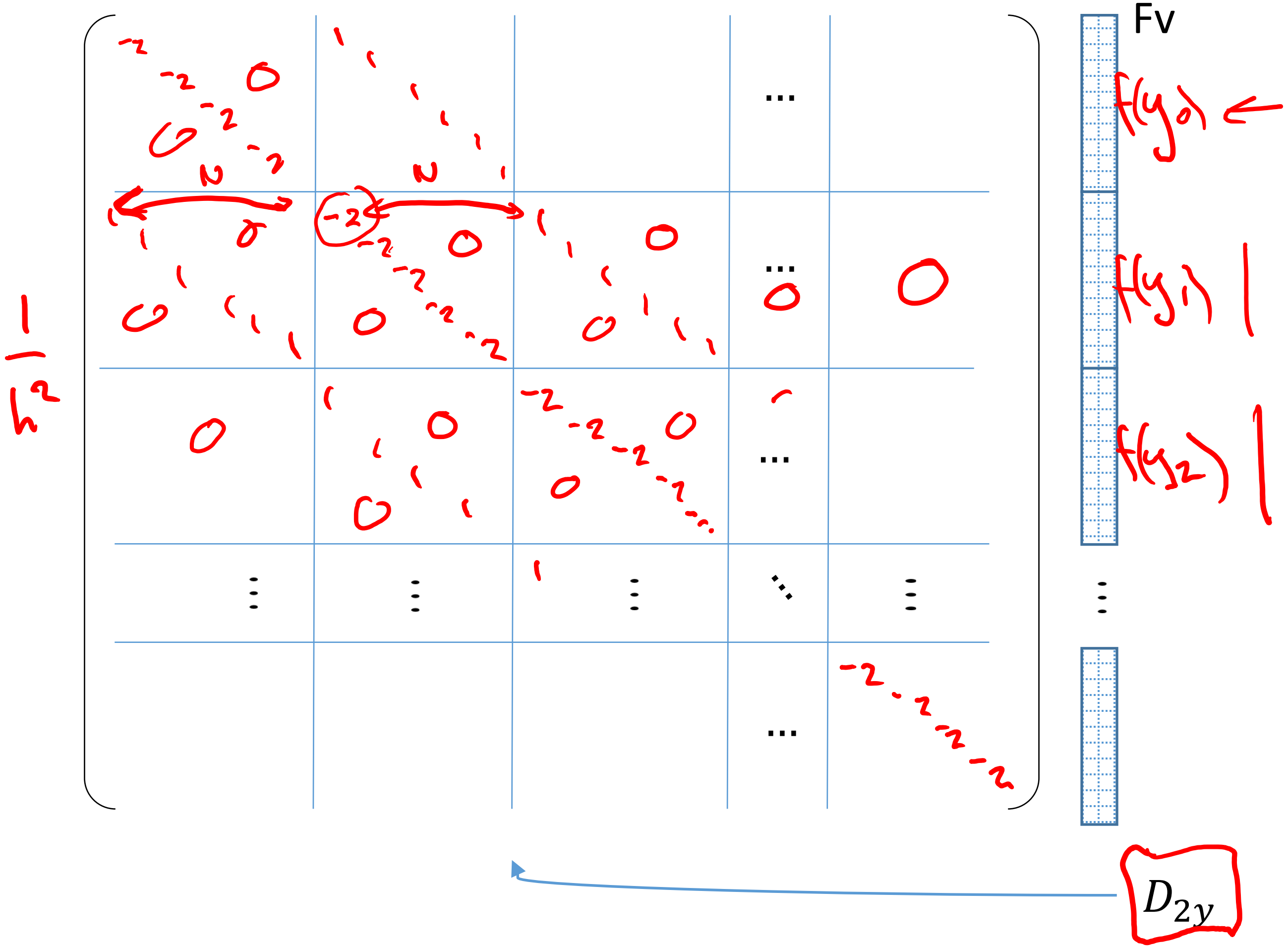
$$D_2 = \frac{1}{h^2}\begin{pmatrix} 1 & -2 & 1 & & & & & & & \\ 1 & -2 & 1 & & & & & & & \\ & & & \ddots & & & & & & \\ & & & 1 & -2 & 1 & & & & \\ & & & & 1 & -2 & 1 & & & \\ & & & & & 1 & -2 & 1 & & \\ & & & & & & & \ddots & & \\ & & & & & & & 1 & -2 & 1 \\ & & & & & & & 1 & -2 & 1 \end{pmatrix}$$



The partial derivative with respect to y

To get the partial derivative with respect to y, we need to interleave the D2 matrix so that it hits the right y values:

$$D_2 = \frac{1}{h^2}\begin{pmatrix} 1 & -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & & \ddots & & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & & \ddots & & \\ & & & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 & 1 \end{pmatrix}$$



$$\left.\frac{d^2 f}{d y^2}\right|_{y_1} \approx \frac{1}{h^2}\left(f(y_0) - 2\underline{f(y_1)} + f(y_2)\right)$$

Forming the Laplacian (and more complicated operators)

General PDEs like the Laplacian can be constructed just by adding matrices:

$$\nabla^2 = \underbrace{\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}} \approx D_{2x} + D_{2y}$$

M×M matrix

M

More complicated derivatives can also be constructed. E.g.

$$\frac{\partial^3}{\partial x^3} + \frac{\partial^4}{\partial y^4} = \quad D_x D_{2x} + D_{2y} D_{2y}$$

$$\underbrace{2x\frac{\partial}{\partial x} + \frac{\partial^2}{\partial x \partial y}}_{L} =$$

We can then create and solve PDEs by forming the matrix equation

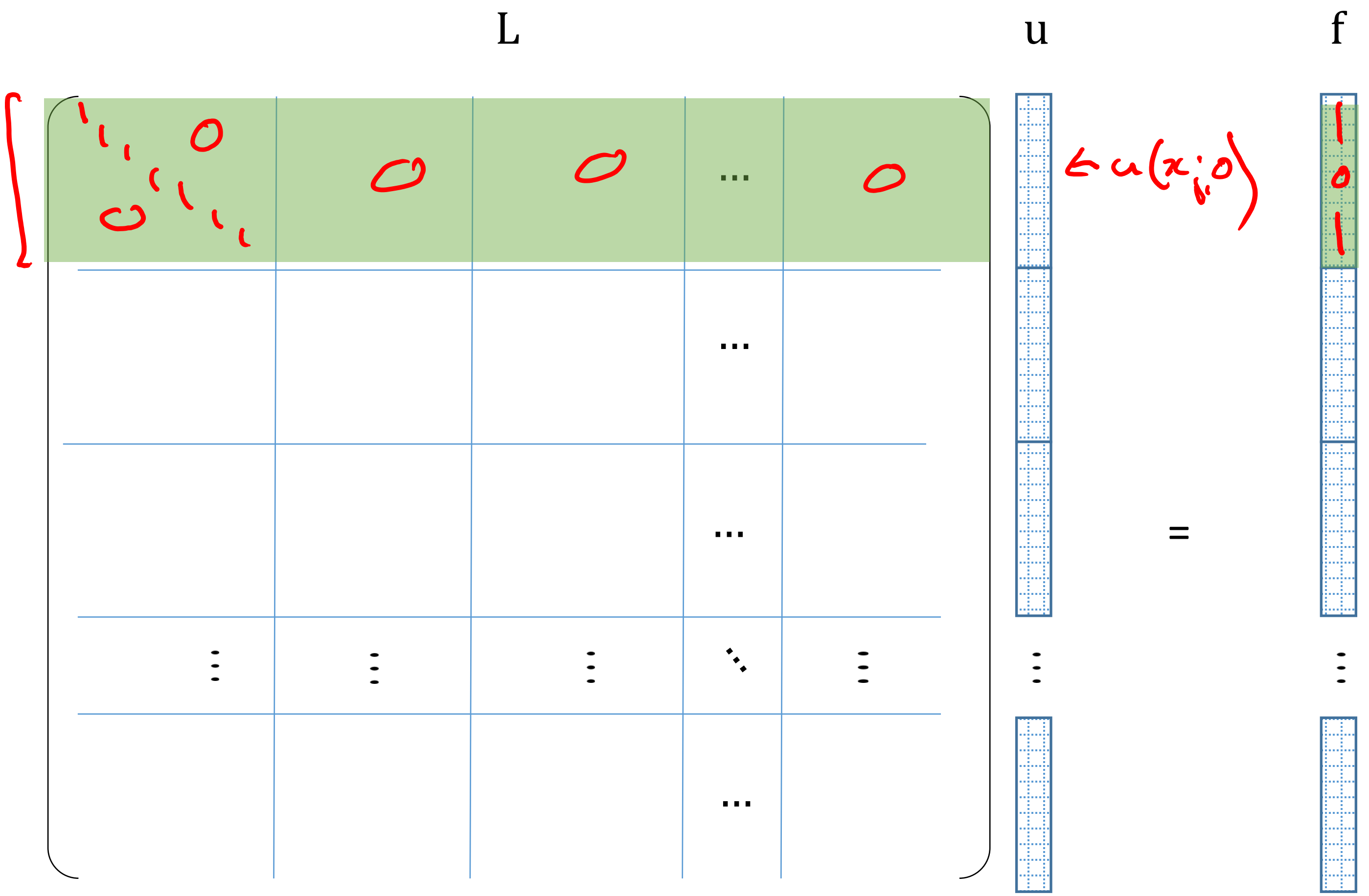
$$\underbrace{L\,u} = f$$

$u = L^{-1} f.$

Boundary conditions can be applied by modifying the resulting matrix
To create equations that are like, for example:

$y=0$

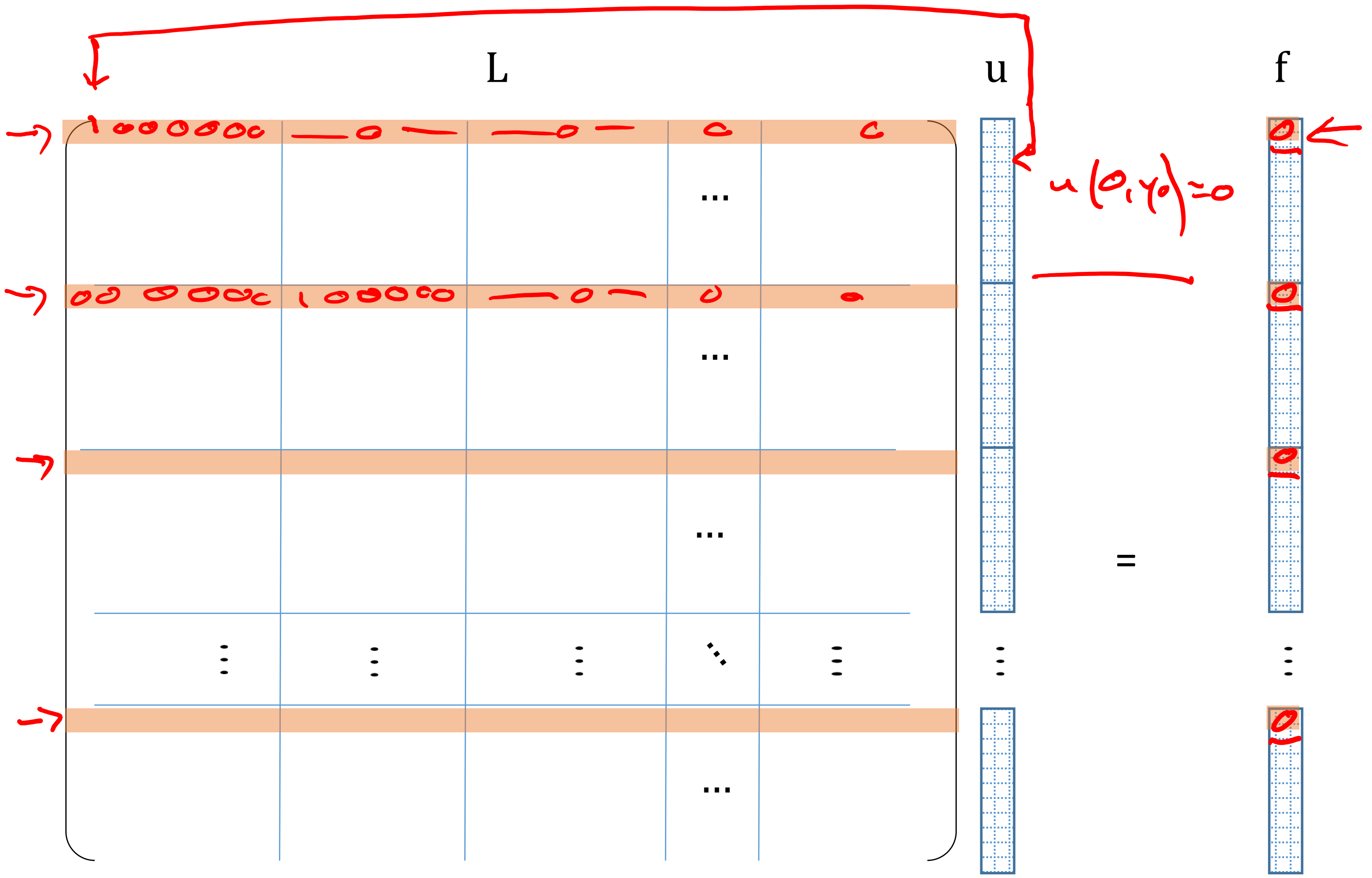
$u(x,0) = 0$:



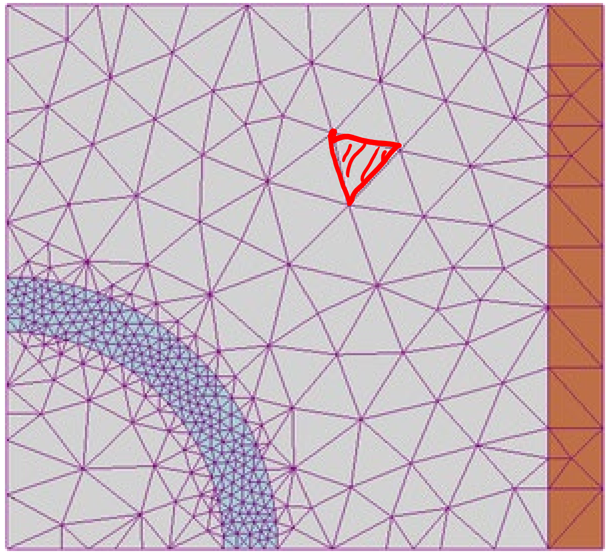
Boundary conditions can be applied by modifying the resulting matrix
To create equations that are like, for example:

$u(0,y) = 0$:

$x=0$



Alternative numerical methods for PDEs:



Finite Element Method
Expand in a mesh of simplexes (i.e. triangles), on which the solution is interpolated using polynomials

The Boundary Element Method
Uses Green's functions on the boundary to construct the solution in an interior domain

Semi-analytical or combined methods
e.g. Crank-Nicholson, Modal methods

The End