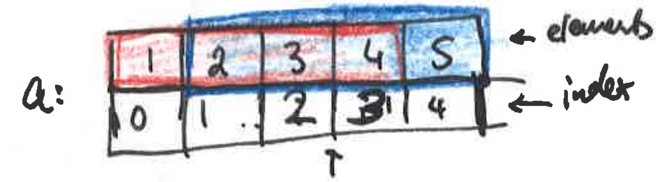
Calculate midpoints m_i of an 1-d array a

Mathematically

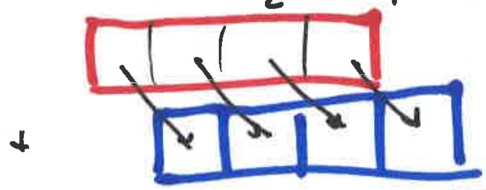
$$m_i = \frac{1}{2}(a_i + a_{i+1})$$



$$\frac{1}{2}(3+4) = \frac{1}{2}\cdot 7 = 3.5$$

With loops (slow)

```
for i in range(len(a)-1):
    m[i] = 0.5*(a[i] + a[i+1])
```



0.5 * 1+2 | 2+3 | 3+4 | 4+5

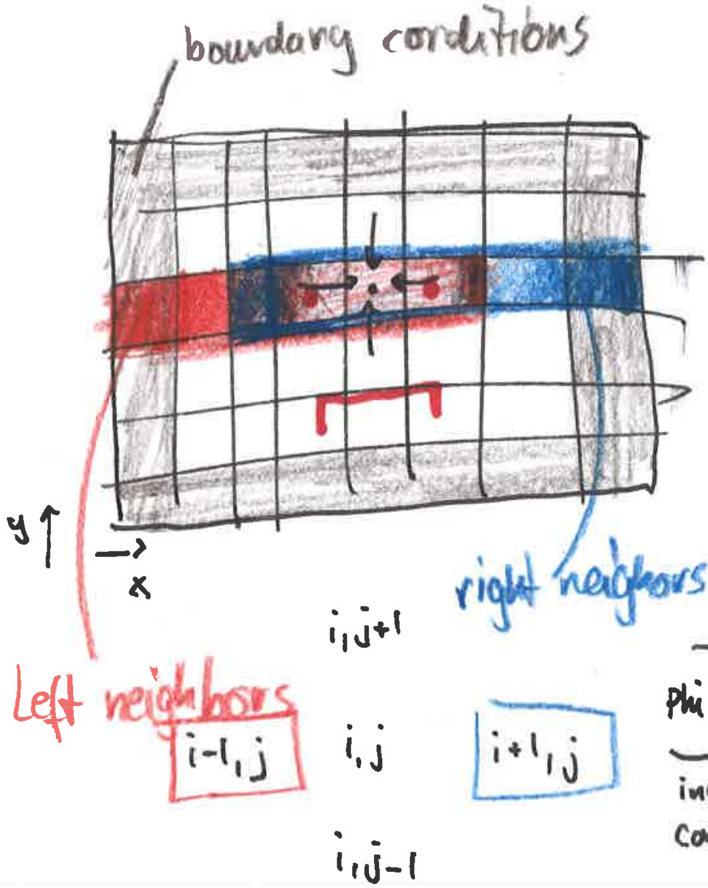
1.5 | 2.5 | 3.5 | 4.5

Numpy array operations

```
m = 0.5*(a[:-1] + a[1:])
```

Jacobi algorithm

$$\phi_{ij} = \frac{1}{4}(\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1})$$



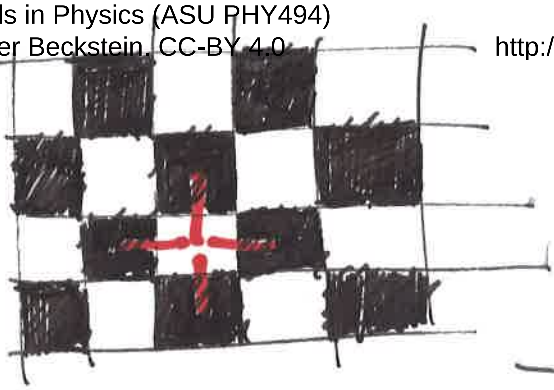
With loops (slow)

```
Phi_new = Phi.copy()
for i in range(1, Phi.shape[0]-1):
   for j in range(1, Phi.shape[1]-1):
      Phi_new[i,j] = 0.25*(Phi[i-1,j]
                   + Phi[i+1,j]
                   + Phi[i,j-1] + Phi[i,j+1])
```

Numpy Arrays

```
Phi[1:-1, 1:-1] = 0.25*(Phi[:-2, 1:-1]     left neighb.
                      + Phi[2:, 1:-1]      right neighbors
                      + Phi[1:-1, :-2]     bottom neighbor
                      + Phi[1:-1, 2:])     top neighbors
```

$\underbrace{\texttt{Phi[1:-1, 1:-1]}}_{\text{inside boundary conditions}}$



Gauss-Seidel-like algorithm in numpy

1) White ('even') lattice sites only require ϕ from black sites.
   Black sites only require data from white sites.

black → white
white → black

black	'odd'
white	'even'

2) Solve black and white sub-lattices separately with the Jacobi algorithm

3) Use output of the black lattice as input for the white solution: In effect part of this iteration's solution is mixed in as in the Gauss-Seidel method.

In the implementation, each sublattice is split into a lattice on even and odd lines in order to be able to use slicing.

```python
def Poisson_Gauss_Seidel_odd_even(Phi, rho, Delta=1.):
    """One update in the Gauss-Seidel algorithm for Poisson's equation on odd or even fields"""

    a = np.pi * Delta**2
    # odd update (uses old even)
    Phi[1:-2:2, 1:-2:2] = 0.25*(Phi[2::2, 1:-2:2] + Phi[0:-2:2, 1:-2:2]  \
                              + Phi[1:-2:2, 2::2] + Phi[1:-2:2, 0:-2:2]) \
                              + a * rho[1:-2:2, 1:-2:2]
    Phi[2:-1:2, 2:-1:2] = 0.25*(Phi[3::2, 2:-1:2] + Phi[1:-2:2, 2:-1:2]  \
                              + Phi[2:-1:2, 3::2] + Phi[2:-1:2, 1:-2:2]) \
                              + a * rho[1:-2:2, 1:-2:2]

    # even update (uses new odd)
    Phi[1:-2:2, 2:-1:2] = 0.25*(Phi[2::2, 2:-1:2] + Phi[0:-2:2, 2:-1:2]  \
                              + Phi[1:-2:2, 3::2] + Phi[1:-2:2, 1:-1:2]) \
                              + a * rho[1:-2:2, 2:-1:2]
    Phi[2:-1:2, 1:-2:2] = 0.25*(Phi[3::2, 1:-2:2] + Phi[1:-2:2, 1:-2:2]  \
                              + Phi[2:-1:2, 2::2] + Phi[2:-1:2, 0:-2:2]) \
                              + a * rho[2:-1:2, 1:-2:2]

    return Phi
```