Modeling of the electrostatic potential of a grid

Donglai Ma, 2020,12

Modeling of the electrostatic potential of a grid

- Background
- 2D Laplace Solver
  - Algorithm
  - Key code
  - Result
- Efield
  - Key code
  - Result
- Ray tracing
  - Algorithm
    - Interpolation
      - Code of interp E field
    - 2nd order Runge-Kutta
    - Code of Ray tracing
  - Ray tracing result
- Reference

Background

Instruments are often required to have outer surfaces at spacecraft ground, so they don't emit or collect photoelectrons and charge up (or cause distortions of the geophysical electric potential/field measurement). Often a grid is placed next to an opening to reduce the fields from internal charged surfaces. Grids also appear in the internal electrostatic design of instruments, in order to help guide particles in appropriate directions, e.g., scattered electrons in time-of-flight instruments or ions exiting from the electrostatic deflection plates. Leakage fields, however, passing through the grid, make the grid appear as having an effective potential, not ground or the intended potential. A grid design is a compromise between the density of lines which affects the particle transmission, and the leakage field that is suppressed enough. Determine a grid's effective potential for a grid of given size/spacing.

2D Laplace Solver

Algorithm

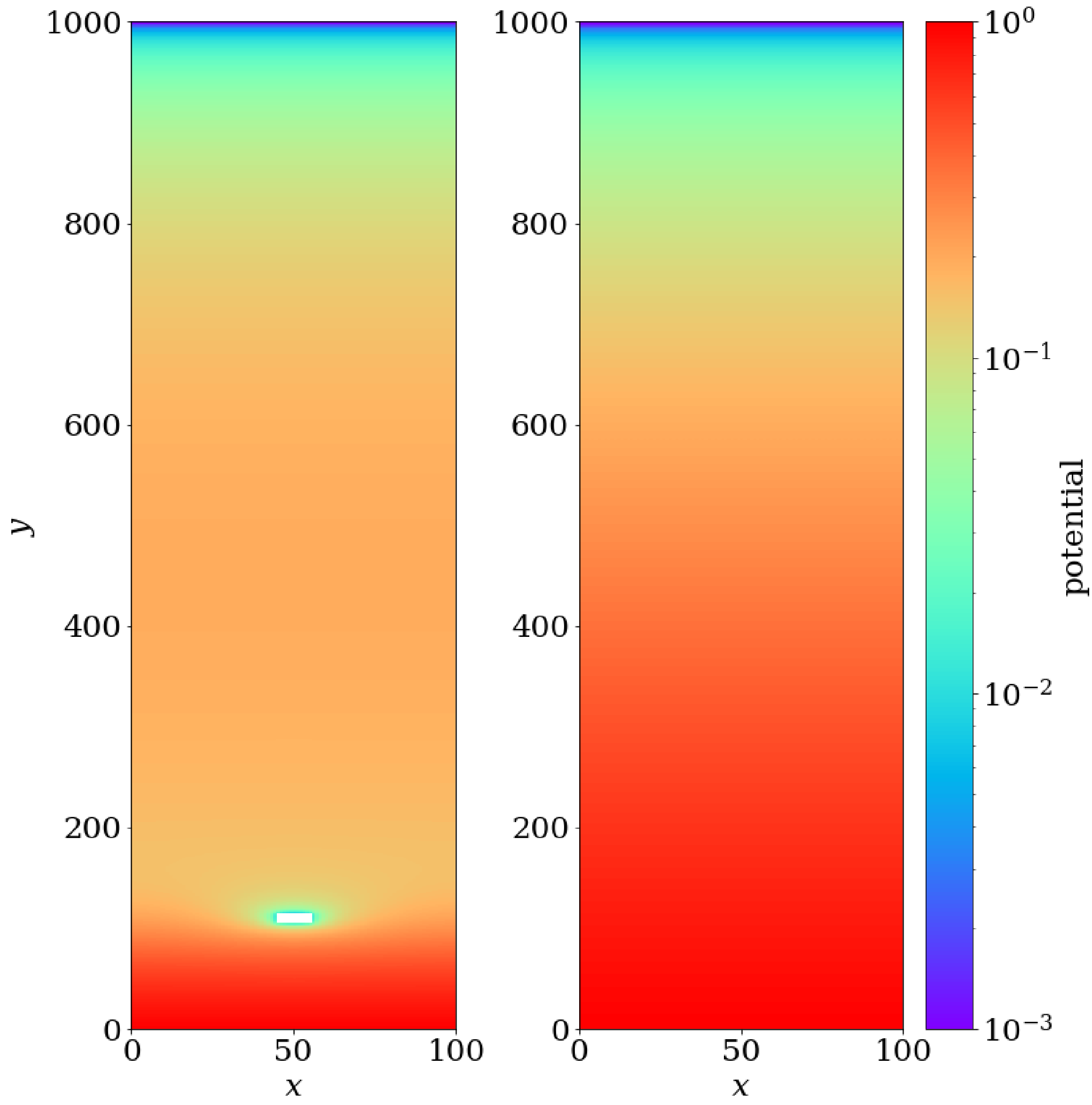
$$U_{i,j} = \frac{U_{i+1,j}+U_{i-1,j}+U_{i,j+1}+U_{i,j-1}}{4}$$

Key code

```
def laplace2d(p, l2_target, ma_wire):
    '''Iteratively solves the Laplace equation using the Jacobi method

    Parameters:
    ----------
    p: 2D array of float
        Initial potential distribution
    l2_target: float
        target for the difference between consecutive solutions

    Returns:
    -------
    p: 2D array of float
        Potential distribution after relaxation
    '''

    l2norm = 1
    pn = np.empty_like(p)
    while l2norm > l2_target:
        pn = p.copy()
        p[1:-1,1:-1] = .25 * (pn[1:-1,2:] + pn[1:-1, :-2] \
                              + pn[2:, 1:-1] + pn[:-2, 1:-1])
```

```
        ##periodic B.C. along x
        p[1:-1, 0] = .25 * (pn[1:-1, 1] + pn[1:-1, -2] \
                            + pn[2:, 0] + pn[:-2, 0])
        p[1:-1, -1] = p[1:-1, 0]
        ##zero potential at the wire
        p[ma_wire] = 0.0

        ##relative l2 norm
        l2norm = L2_error(p, pn)

    return p
```

Result



Efield

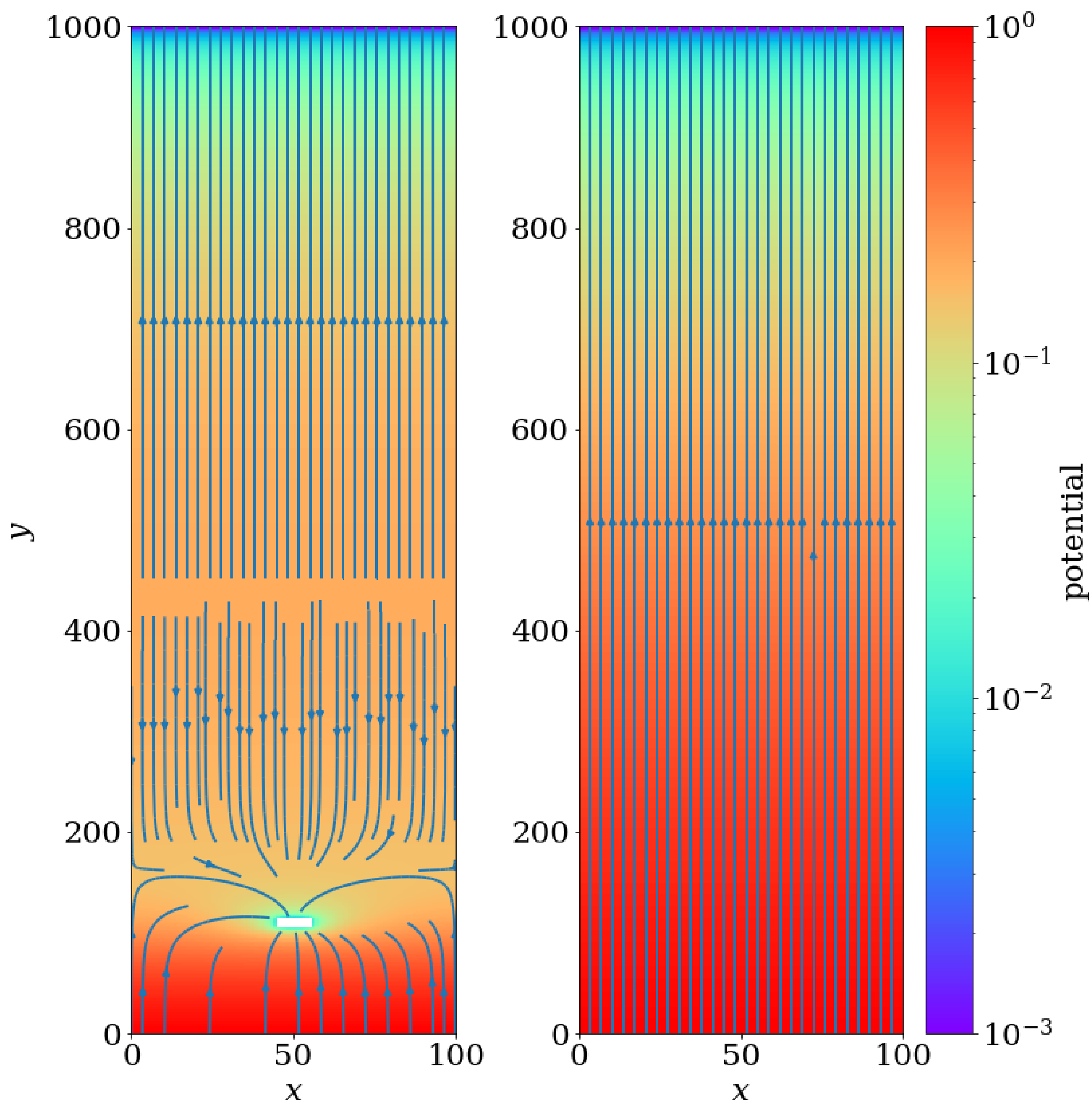
$E_1 = -(V2 - V0)/2$

for Ey boundary, I use E0 = E1, E1000 = E999

Key code

```
def getEfield(pot):
    """
    Get the field from position
    """
    # Calculate the E field
    # ie . E1 = -(V2 - V0)/2

    Ex = -(np.hstack((pot[:,1:],pot[:,0:1])) - np.hstack((pot[:,-1:],pot[:,0:-1])))/2
    # For Ey
    Ey = -(pot[2:,:]- pot[:-2,:])/2

    # Here without Ey boundary, E0 = E1 E1000 =E999
    #Add Ey boundary,
    Ey_0 = Ey[0:1,:]
    Ey_1000 = Ey[-1:,:]
    Ey = np.vstack((Ey_0,Ey))
    Ey = np.vstack((Ey,Ey_1000))

    return Ex,Ey
```

Result

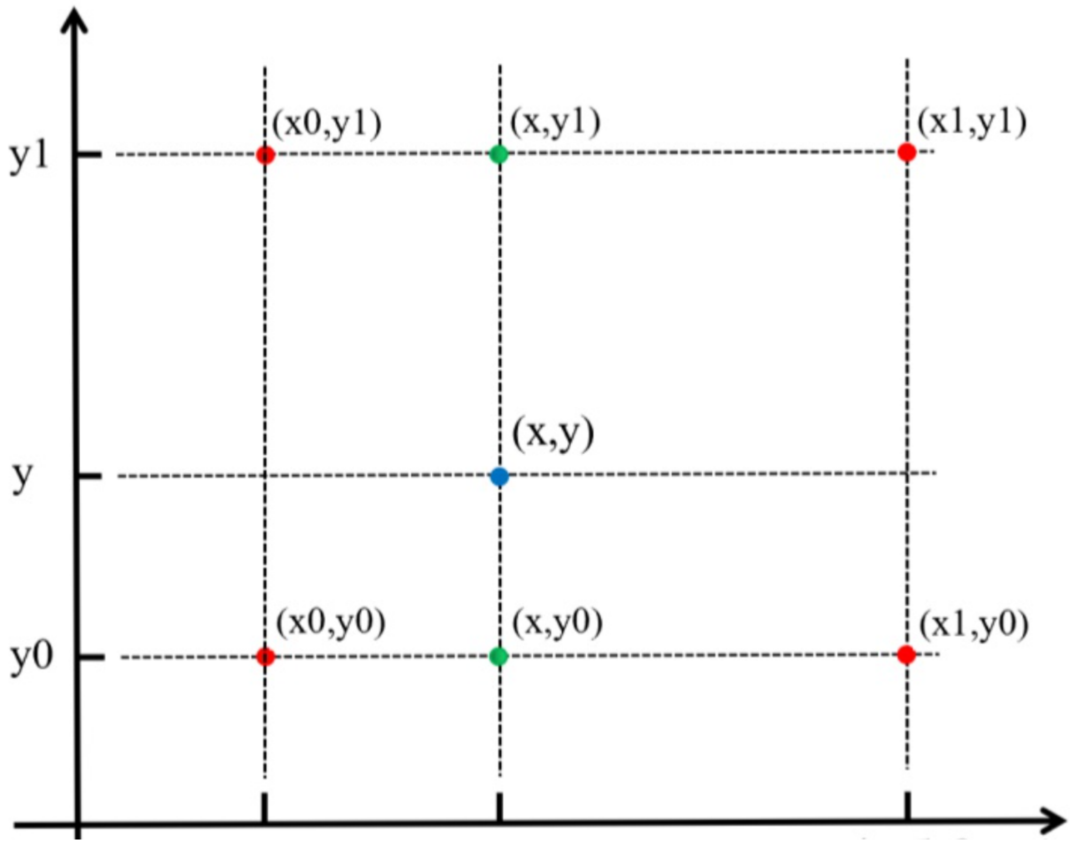


Ray tracing

Algorithm

Interpolation

When calculating the E field of particle , use linear interpolation



$$f(x,y)=\frac{(y_1-y)(x_1-x)}{(y_1-y_0)(x_1-x_0)}f\left(x_0,y_0\right)+\frac{(y_1-y)(x-x_0)}{(y_1-y_0)(x_1-x_0)}f\left(x_1,y_0\right)+\frac{(y-y_0)(x_1-x)}{(y_1-y_0)(x_1-x_0)}f\left(x_0,y_1\right)+\frac{(y-y_0)(x-x_0)}{(y_1-y_0)(x_1-x_0)}f\left(x_1,y_1\right)$$

So that E field is :

$$Ex(x,y)=\left(y_1-y\right)\left(x_1-x\right)Ex\left(x_0,y_0\right)+\left(y_1-y\right)\left(x-x_0\right)Ex\left(x_1,y_0\right)+\left(y-y_0\right)\left(x_1-x\right)Ex\left(x_0,y_1\right)+\left(y-y_0\right)\left(x-x_0\right)Ex\left(x_1,y_1\right)$$

and

$$Ey(x,y)=\left(y_1-y\right)\left(x_1-x\right)Ey\left(x_0,y_0\right)+\left(y_1-y\right)\left(x-x_0\right)Ey\left(x_1,y_0\right)+\left(y-y_0\right)\left(x_1-x\right)Ey\left(x_0,y_1\right)+\left(y-y_0\right)\left(x-x_0\right)Ey\left(x_1,y_1\right)$$

Code of interp E field

```
def interpfield(x,y,Ex,Ey):
    """
    interp the E field
    (position is not on grid when do tracing)
    Use inter linear
    f(x,y)

    """
    xx = x%100
    yy = y
    x0 = int(xx)
    x1 = int(xx) + 1
    y0 = int(yy)
    y1 = int(yy) + 1

    Ex_p = (y1-yy)*(x1-xx)*Ex[y0,x0] + (y1-yy)*(xx-x0)*Ex[y0,x1] + (yy-y0)*(x1-xx)*Ex[y1,x0] + (yy-y0)*
(xx-x0)*Ex[y1,x1]
    Ey_p = (y1-yy)*(x1-xx)*Ey[y0,x0] + (y1-yy)*(xx-x0)*Ey[y0,x1] + (yy-y0)*(x1-xx)*Ey[y1,x0] + (yy-y0)*
(xx-x0)*Ey[y1,x1]
    return Ex_p,Ey_p
```

2nd order Runge-Kutta

$$
\begin{aligned}
\Delta X_1 &= \Delta t\left(V^t\right) & \Delta V_1 &= \Delta t A\left(X^t, V^t\right) \\
\Delta X_2 &= \Delta t\left(V^t + \Delta V_1/2\right) & \Delta V_2 &= \Delta t A\left(X^t + \Delta X_1/2, V^t + \Delta V_1/2\right)
\end{aligned}
$$

$$
\begin{aligned}
X^{t+1} &= X^t + \left(\Delta X_1 + \Delta X_2\right)/2 \\
V^{t+1} &= V^t + \left(\Delta V_1 + \Delta V_2\right)/2
\end{aligned}
$$

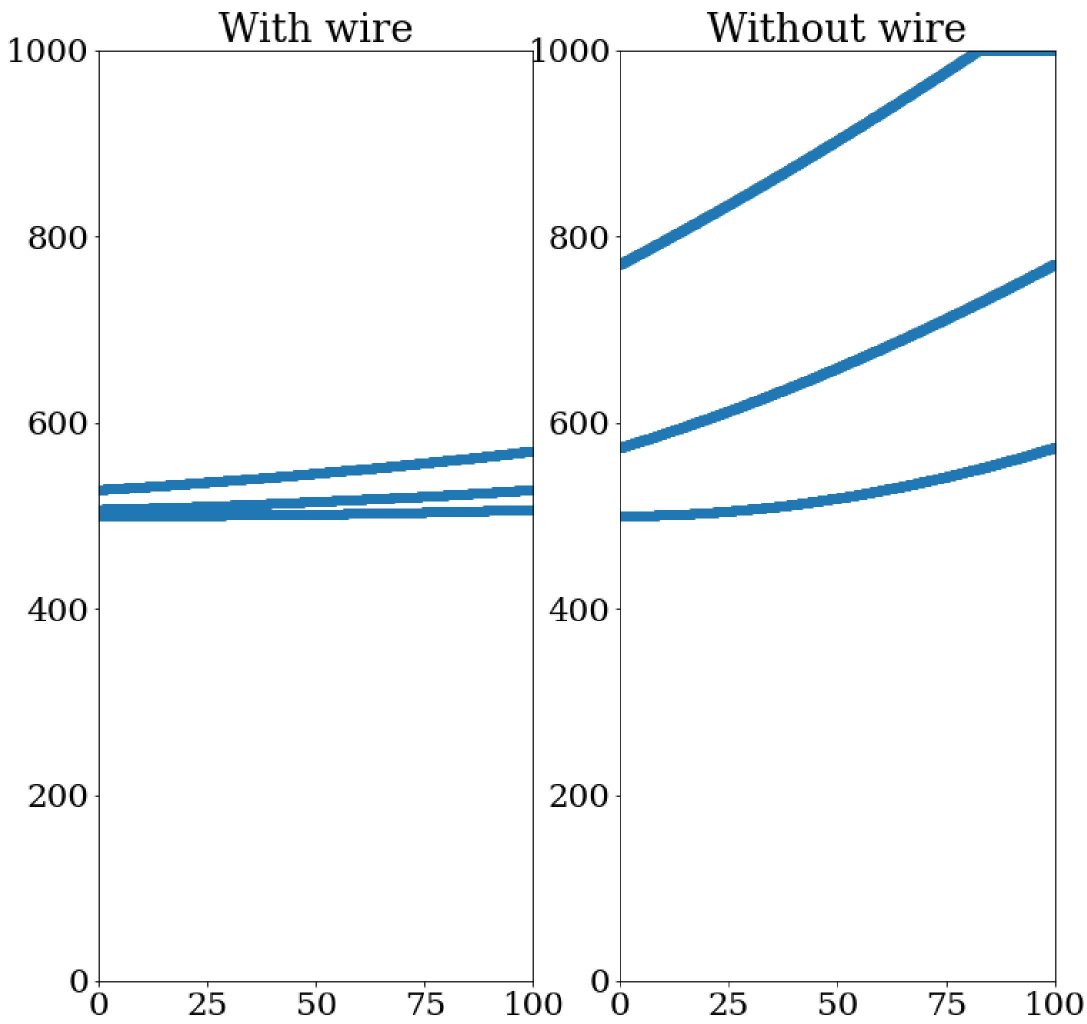
Code of Ray tracing

```
def trace(x0,y0,vx0,vy0,pot,steps = 1000,t=0.01,alpha = 1 ):
    """
    Get the particle trace
    """
    # trace save the information of position and velocity
    x_trace = np.zeros(steps)
    y_trace = np.zeros(steps)
    vx_trace = np.zeros(steps)
    vy_trace = np.zeros(steps)


    x_trace[0] = x0
    y_trace[0] = y0
    vx_trace[0] = vx0
    vy_trace[0] = vy0

    Ex,Ey = getEfield(pot)

    # p means particle
    Ex_p,Ey_p = interpfield(x_trace[0],y_trace[0],Ex,Ey)
    for i in range(1,steps,1):
        # Use RK2
        vxt,vyt = vx_trace[i-1],vy_trace[i-1]
        xt,yt = x_trace[i-1],y_trace[i-1]


        Ex_p,Ey_p = interpfield(xt,yt,Ex,Ey)

        Dx1,Dy1 = vxt*t,vyt*t

        Dvx1,Dvy1 = t* alpha*Ex_p ,t* alpha*Ey_p

        Dx2,Dy2 = t*(vxt + Dvx1/2), t*(vyt + Dvy1/2)

        Ex_p1,Ey_p1 = interpfield(xt + Dx1/2,yt + Dy1/2,Ex,Ey)
        Dvx2,Dvy2 = t*alpha*Ex_p1,t*alpha*Ey_p1
        xt1,yt1 = xt +(Dx1 + Dx2)/2,yt +(Dy1 + Dy2)/2
        vxt1,vyt1 = vxt +(Dvx1 + Dvx2)/2,vyt +(Dvy1 + Dvy2)/2

        # Save the trace
        if yt1>1000:
            yt1 = 1000
        if yt1 < 0:
            yt1 = 0
        x_trace[i],y_trace[i] = xt1%100,yt1

        vx_trace[i],vy_trace[i] = vxt1,vyt1




    return x_trace,y_trace
```

Ray tracing result

x0 = 0
y0 = 500
vx0 = 10
vy0 = 0

alpha = 1800(I did not set the parameters rigorously)



Reference

Wüest, Martin, David S. Evans, and Rudolf von Steiger, eds. *Calibration of particle instruments in space physics*. International Space Science Institute, 2007.