forwardScattering_shapes

May 2, 2024

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from scipy.special import hankel2, jv

def generate_A(coords_s:tuple[np.ndarray],coords_b:tuple[np.ndarray]) -> np.
 ↪ndarray:
    # number of sources and boundary elements
    xs, ys = coords_s
    xb, yb = coords_b
    N = len(xs)
    M = len(xb)
    # resulting matrix
    A = np.zeros((M,N), dtype=complex)
    # for each line
    for i in range(M):
        # coordinate of boundary element
        xi = xb[i]
        yi = yb[i]
        r2 = (xs-xi)**2 + (ys-yi)**2
        A[i] = hankel2(0,np.sqrt(r2))
    return A

def inc_field(coords:tuple[np.ndarray], phi_inc:float=0) -> np.ndarray:
    x,y = coords
    return np.exp(-1j*x*np.cos(phi_inc) - 1j*y*np.sin(phi_inc))

def inc_point_source(coords:tuple[np.ndarray],r:float,phi_inc:float=0) -> np.
 ↪ndarray:
    xs = r*np.cos(phi_inc)
    ys = r*np.sin(phi_inc)
    x,y = coords
    r2 = (x-xs)**2 + (y-ys)**2
def inc_field_noisy(coords:tuple[np.ndarray],phi_std:float, phi_inc:float=0) ->␣
 ↪np.ndarray:
    x,y = coords
```

```python
    return np.exp(-1j*x*np.cos(phi_inc) - 1j*y*np.sin(phi_inc) + 1j*np.random.
↪normal(0,phi_std,len(x)))

def generate_A(coords_s:tuple[np.ndarray],coords_b:tuple[np.ndarray]) -> np.
↪ndarray:
    # number of sources and boundary elements
    xs, ys = coords_s
    xb, yb = coords_b
    N = len(xs)
    M = len(xb)
    # resulting matrix
    A = np.zeros((M,N), dtype=complex)
    # for each line
    for i in range(M):
        # coordinate of boundary element
        xi = xb[i]
        yi = yb[i]
        r2 = (xs-xi)**2 + (ys-yi)**2
        A[i] = hankel2(0,np.sqrt(r2))
    return A

def sca_field(c:np.ndarray, coords_s:tuple[np.ndarray], coords_m:tuple[np.
↪ndarray]) -> np.ndarray:
    xs, ys = coords_s
    xm, ym = coords_m
    N = len(xs)
    M = len(xm)
    sca = np.zeros(M, dtype=complex)  # Initialize scattered field vector
    for i in range(N):
        xi = xs[i]
        yi = ys[i]
        r2 = (xm-xi)**2 + (ym-yi)**2
        sca += c[i]*hankel2(0,np.sqrt(r2))
    return sca

def theo_field(R_b:float, coords_m:tuple[np.ndarray],phi_inc:float = 0) -> np.
↪ndarray:
    xm, ym = coords_m
    phim = np.arctan2(ym, xm)
    # precision of theoretical estimate
    N = 200
    M = len(xm)
    theo = np.zeros(M, dtype=complex)

    for i in np.arange(-N,N,1):
        r = np.sqrt(xm**2 + ym**2)
        c = -1j**(-i) * jv(i,R_b)/hankel2(i,R_b)
```

```
        theo += c*hankel2(i,r)*np.exp(1j*i*(phim - phi_inc))
    return theo

def coords2deg(coords:tuple[np.ndarray]) -> np.ndarray:
    x, y = coords
    return np.rad2deg(np.arctan2(y,x))
```

[ ]:
```
def ellypse_coords(rx: float, ry: float, N:int) -> tuple[np.ndarray]:
    phi_stop = + np.pi - 2*np.pi/N
    phi = np.linspace(- np.pi, phi_stop, N)
    x = rx*np.cos(phi)
    y = ry*np.sin(phi)
    return x,y

def rectangle_coords(lx: float, ly: float, N: int) -> np.ndarray:
    N = 4*(N//4)
    x = np.zeros(N)
    y = np.zeros(N)

    # Points on the top side
    x[:N//4] = np.linspace(- lx/2,lx/2, N//4)
    y[:N//4] = ly/2
    # Points on the right side
    x[N//4:N//2] = lx/2
    y[N//4:N//2] = np.linspace(ly/2,- ly/2, N//4)
    # Points on the bottom side
    x[N//2:3*N//4] = np.linspace(lx/2,- lx/2, N//4)
    y[N//2:3*N//4] = - ly/2
    # Points on the left side
    x[3*N//4:] = - lx/2
    y[3*N//4:] = np.linspace(- ly/2,ly/2, N//4)

    return x,y

def kite_coords(l: float, N: int) -> np.ndarray:
    phi_stop = np.pi - 2*np.pi/N
    phi = np.linspace(- np.pi , phi_stop, N)
    x = 1.5*l * (np.cos(phi) + 0.65 * np.cos(2*phi) - 0.65) + 0.25*l
    y = 1.5*l * np.sin(phi)

    return x,y

def shape_coords(l: float, N:int, shape:int) -> np.ndarray:
    if shape == 0:
        return ellypse_coords(l,l,N)
    if shape == 1:
        return ellypse_coords(l,2*l,N)
```

```
    if shape == 2:
        return rectangle_coords(2*l,2*l,N)
    if shape == 3:
        return rectangle_coords(2*l,3*l,N)
    if shape == 4:
        return kite_coords(l,N)
```

[ ]:
```
## PARAMETERS ##
# define  normalized radius of sources, boundary, measurements and mesh
R_b = 3
R_s = 0.95*R_b
R_m = 5
R_s *= 2*np.pi
R_b *= 2*np.pi
R_m *= 2*np.pi
# define number of points
N_s = 100
N_b = 500
N_m = 500
# define starting angle
phi_inc = 0
# number of shapes
N_shape = 5
```

[ ]:
```
def generate_A(coords_s:tuple[np.ndarray],coords_b:tuple[np.ndarray]) -> np.
 ↪ndarray:
    # number of sources and boundary elements
    xs, ys = coords_s
    xb, yb = coords_b
    N = len(xs)
    M = len(xb)
    # resulting matrix
    A = np.zeros((M,N), dtype=complex)
    # for each line
    for i in range(M):
        # coordinate of boundary element
        xi = xb[i]
        yi = yb[i]
        r2 = (xs-xi)**2 + (ys-yi)**2
        A[i] = hankel2(0,np.sqrt(r2))
    return A

def sca_field(c:np.ndarray, coords_s:tuple[np.ndarray], coords_m:tuple[np.
 ↪ndarray]) -> np.ndarray:
    xs, ys = coords_s
    xm, ym = coords_m
```

```python
    N = len(xs)
    M = len(xm)
    sca = np.zeros(M, dtype=complex)  # Initialize scattered field vector
    for i in range(N):
        xi = xs[i]
        yi = ys[i]
        r2 = (xm-xi)**2 + (ym-yi)**2
        sca += c[i]*hankel2(0,np.sqrt(r2))
    return sca

def theo_field(R_b:float, coords_m:tuple[np.ndarray],phi_inc:float = 0) -> np.
 ↪ndarray:
    xm, ym = coords_m
    phim = np.arctan2(ym, xm)
    # precision of theoretical estimate
    N = 200
    M = len(xm)
    theo = np.zeros(M, dtype=complex)

    for i in np.arange(-N,N,1):
        r = np.sqrt(xm**2 + ym**2)
        c = -1j**(-i) * jv(i,R_b)/hankel2(i,R_b)
        theo += c*hankel2(i,r)*np.exp(1j*i*(phim - phi_inc))
    return theo
```

[ ]:
```python
## parameter swipe of N##
N_b_s = 500

es = np.zeros((N_shape,N_b_s - 1))
ks = np.zeros((N_shape,N_b_s - 1))
vals = range(1,N_b_s)

for shape in range(0,N_shape):
    print(shape)
    coords_b = shape_coords(R_b,N_b_s,shape)
    inc_b = inc_field(coords_b,phi_inc)
    for n_s in vals:
        coords_s = ellypse_coords(R_s, R_s, n_s)
        A = generate_A(coords_s,coords_b)
        c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
        sca_b = sca_field(c,coords_s,coords_b)
        e = np.mean(np.abs(sca_b + inc_b)**2)
        es[shape][n_s - 1] = e
        ks[shape][n_s - 1] = np.linalg.cond(A)
```

[ ]:
```
for shape in range(0,N_shape):
    plt.plot(vals,es[shape])
plt.axvline(x = N_b_s, color='b',linestyle="--")
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite',r'$M$'])
plt.title(r'MSE as a function of sources', fontsize=15)
plt.xlabel(r'Number of auxilary sources ($N$)', fontsize=15)
plt.ylabel(r'MSE at PEC boundary $\langle|E/E_0|^2\rangle$', fontsize=15)
plt.savefig("./E_S.pdf",bbox_inches='tight')
plt.show()

for shape in range(0,N_shape):
    plt.plot(vals,ks[shape])
plt.axvline(x = N_b_s, color='b',linestyle="--")
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite',r'$M$'])
plt.yscale("log")
plt.title(r'Conditionning number as a function of sources', fontsize=15)
plt.xlabel(r'Number of auxilary sources ($N$)', fontsize=15)
plt.ylabel(r'Conditionning number $\kappa (A)$', fontsize=15)
plt.savefig("./K_S.pdf",bbox_inches='tight')
```

[ ]:
```
valb = range(N_s,N_b)

valb_min = valb[0]
eb = np.zeros((N_shape,len(valb)))
kb = np.zeros((N_shape,len(valb)))
coords_s = ellypse_coords(R_s, R_s, N_s)


for shape in range(0,N_shape):
    print(shape)
    for n_b in valb:
        coords_b = shape_coords(R_b, n_b, shape)
        A = generate_A(coords_s,coords_b)
        inc_b = inc_field(coords_b,phi_inc)
        c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
        sca_b = sca_field(c,coords_s,coords_b)
        e = np.mean(np.abs(sca_b + inc_b)**2)
        eb[shape][n_b - valb_min] = e
        kb[shape][n_b - valb_min] = np.linalg.cond(A)
```

[ ]:
```
for shape in range(0,N_shape):
    plt.plot(valb,eb[shape])
plt.axvline(x = N_s, color='b',linestyle="--")
plt.axvline(x = 3*N_s, color='g',linestyle="--")
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite',r'$N$',r'3N'])
plt.title(r'MSE as a function of boundary points', fontsize=15)
plt.xlabel(r'Number of boundary points ($M$)', fontsize=15)
```

```python
plt.ylabel(r'MSE at PEC boundary $\langle|E/E_0|^2\rangle$', fontsize=15)
plt.savefig("./E_B.pdf",bbox_inches='tight')
plt.show()

for shape in range(0,N_shape):
    plt.plot(valb,kb[shape])
plt.axvline(x = N_s, color='b',linestyle="--")
plt.axvline(x = 3*N_s, color='g',linestyle="--")
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite',r'$N$',r'3N'])
plt.yscale("log")
plt.title(r'Conditionning number as a function of boundary points', fontsize=15)
plt.xlabel(r'Number of boundary points ($M$)', fontsize=15)
plt.ylabel(r'Conditionning number $\kappa (A)$', fontsize=15)
plt.savefig("./K_B.pdf",bbox_inches='tight')
```

[ ]:
```python
## parameter swipe of R_s##

N_test = 1000
valr = np.linspace(0,R_b,N_test)[:-1] #remove singularity
valr_min = valr[0]
er = np.zeros((N_shape,len(valr)))
kr = np.zeros((N_shape,len(valr)))
coords_s = ellypse_coords(R_s, R_s, N_s)

for shape in range(0,N_shape):
    print(shape)
    coords_b = shape_coords(R_b, N_b,shape)
    inc_b = inc_field(coords_b,phi_inc)
    for i in range(len(valr)):
        r_s = valr[i]
        coords_s = ellypse_coords(r_s,r_s, N_s)
        A = generate_A(coords_s,coords_b)
        c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
        sca_b = sca_field(c,coords_s,coords_b)
        e = np.mean(np.abs(sca_b + inc_b)**2)
        er[shape][i] = e
        kr[shape][i] = np.linalg.cond(A)
```

[ ]:
```python
for shape in range(0,N_shape):
    plt.plot(valr/R_b,er[shape])
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite'])
plt.xlabel(r'Distance of source points $R_s/R_\Omega$', fontsize=15)
plt.ylabel(r'MSE at PEC boundary $\langle|E/E_0|^2\rangle$', fontsize=15)
plt.title(r'MSE as a function of source radius', fontsize=15)
plt.savefig("./E_R.pdf",bbox_inches='tight')
plt.show()
```

```
for shape in range(0,N_shape):
    plt.plot(valr/R_b,kr[shape])
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite'])
plt.yscale("log")
plt.xlabel(r'Distance of source points $R_s/R_\Omega$', fontsize=15)
plt.ylabel(r'Conditionning number $\kappa (A)$', fontsize=15)
plt.title(r'Conditionning number as a function of source radius', fontsize=15)
plt.savefig("./K_R.pdf",bbox_inches='tight')
```

[ ]:
```
## parameter swipe of phi_i##
N_test = 1000
valphi = np.linspace(0,2*np.pi,N_test)

ephi = np.zeros((N_shape,len(valphi)))
coords_s = ellypse_coords(R_s, R_s, N_s)

for shape in range(0,N_shape):
    print(shape)
    coords_b = shape_coords(R_b, N_b,shape)
    A = generate_A(coords_s,coords_b)
    for i in range(len(valphi)):
        phi = valphi[i]
        inc_b = inc_field(coords_b,phi)
        c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
        sca_b = sca_field(c,coords_s,coords_b)
        e = np.mean(np.abs(sca_b + inc_b)**2)
        ephi[shape][i] = e
```

[ ]:
```
for shape in range(0,N_shape):
    plt.polar(valphi,ephi[shape])
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite'])
plt.title(r'MSE as a function of incidence angle $\varphi^i$', fontsize=15)
plt.savefig("./E_phi.pdf",bbox_inches='tight')
plt.show()
```

[ ]:
```
## parameter swipe of source_point##

N_test = 1000
valp = np.linspace(R_s,10*R_s,N_test)

ep = np.zeros((N_shape,len(valp)))
coords_s = ellypse_coords(R_s, R_s, N_s)

for shape in range(0,N_shape):
    print(shape)
    coords_b = shape_coords(R_b, N_b,shape)
    A = generate_A(coords_s,coords_b)
```

```python
    for i in range(len(valp)):
        r_ps = valp[i]
        inc_b = inc_point_source(coords_b,r_ps)
        c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
        sca_b = sca_field(c,coords_s,coords_b)
        e = np.mean(np.abs(sca_b/inc_b + 1)**2)
        ep[shape][i] = e
```

[ ]:
```python
for shape in range(0,N_shape):
    plt.plot(valp/R_s,ep[shape])
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite'])
#plt.yscale("log")
plt.xlabel(r'Distance of source points $R_{inc}/R_\Omega$', fontsize=15)
plt.ylabel(r'MSE at PEC boundary $\langle|E/E_0|^2\rangle$', fontsize=15)
plt.title(r'MSE as a function of incident point source radius', fontsize=15)
plt.savefig("./E_P.pdf",bbox_inches='tight')
```

[ ]:
```python
## parameter swipe of noisy source##
N_test = 10
valnoise = np.linspace(0,np.pi/10,N_test)

enoise = np.zeros((N_shape,len(valnoise)))
coords_s = ellypse_coords(R_s, R_s, N_s)

N_shape = 1
for shape in range(0,N_shape):
    print(shape)
    coords_b = shape_coords(R_b, N_b,shape)
    A = generate_A(coords_s,coords_b)
    for i in range(len(valnoise)):
        phi_std = valp[i]
        for j in range(100):
            inc_b = inc_field_noisy(coords_b,phi_std)
            c, residuals, _, _ = np.linalg.lstsq(A, -inc_b, rcond=0.01)
            sca_b = sca_field(c,coords_s,coords_b)
            inc_non_noisy = inc_field(coords_b)
            e = np.mean(np.abs(sca_b/inc_non_noisy + 1)**2)
            enoise[shape][i] += e/10
```

[ ]:
```python
for shape in range(0,N_shape):
    plt.plot(np.rad2deg(valnoise),enoise[shape])
plt.legend([r'Circle',r'Ellypse',r'Square',r'Rectangle',r'Kite'])
#plt.yscale("log")
plt.xlabel(r'phase noise $\sigma(\varphi^{i}) \;$ (°)', fontsize=15)
plt.ylabel(r'MSE at PEC boundary $\langle|E/E_0|^2\rangle$', fontsize=15)
plt.title(r'MSE as a function of incident point source radius', fontsize=15)
plt.savefig("./E_P.pdf",bbox_inches='tight')
```