mml

April 29, 2024

1 Radial Basis Function

1.1 Basics

Jede lineare PDE zweiter Ordnung lässt sich in der Form

$$\sum_{ij}^{n} (A)_{ij} \frac{\partial^2 u(x)}{\partial x_i \partial x_j} + \sum_{i}^{n} b_i \frac{\partial u(x)}{\partial x_i} + cu(x) + d = f(x)$$

ausdrücken. Anhand der Matrix A können diese weiter klassifiziert werden:
elliptisch, falls A positiv oder negativ definit ist, i.e. Defekt $d = 0$ und Trägheitsindex $t = 0$
hyperbolisch, falls $d = 0$ und $t = 1$ oder $t = n - 1$
ultrahyperbolisch, falls $d = 0$ und $1 < t < n - 1$
parabolisch, falls A ausgeartet ist, i.e. $d > 0$.

Der Defekt d und Trägheitsindex t sind wie folgt für

$$\text{diag}(A) = (\lambda_1, ..., \lambda_n)$$

definiert:
$t =$ Anzahl der $j \in \{1, ..., n\}$ mit $\lambda_j < 0$
$d =$ Anzahl der $j \in \{1, ..., n\}$ mit $\lambda_j = 0$

1.2 Poisson Eq. Example (u_5 Larsson, Fornberg)

$$\Delta u(x) = f(x) \quad \text{in } \Omega$$
$$u = g(x) \quad \text{on } \partial\Omega$$

Wähle Beispiel u_5 als bekannte Lösung:

$$u_5(x, y) = \sin\left(\pi(x^2 + y^2)\right)$$

Entsprechend betrachten wir $\dim = 2$ und Ω die Einheitsscheibe. Und es gilt

$$\Delta u_5(x, y) = -4\pi(\pi(x^2 + y^2)\sin\left(\pi(x^2 + y^2)\right) - \cos\left(\pi(x^2 + y^2)\right))$$

1.2.1 Approximation mit Methode 1 (Unsymmetrisch):

$$s(x, \epsilon) = \sum_{j=1}^{N} \lambda_j \phi(||x - x_j||, \epsilon)$$

Dabei wird für $\phi(r, \epsilon)$ die Multiquadric RBF gewählt:

$$\phi(r, \epsilon) = \sqrt{1 + (\epsilon r)^2}$$

1.2.2 Exkurs Polarkoordinaten

Jeder Punkt (x, y) lässt sich auch eindeutig durch seinen Abstand r vom Ursprung und den Winkel θ zwischen der Geraden vom Ursprung zum Punkt (x, y) und der x-Achse darstellen. Diese Koordinatentransformation wird durch

$$x = r\cos(\theta)$$

$$y = r\sin(\theta)$$

beschrieben. Das heißt eine Funktion $u(x, y)$ kann auch als $u(x(r, \theta), y(r, \theta))$ beschrieben werden und mit der Kettenregel folgt

$$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial r}$$

$$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x}\cos(\theta) + \frac{\partial u}{\partial y}\sin(\theta)$$

und

$$\frac{\partial^2 u}{\partial r^2} = \frac{\partial^2 u}{\partial x^2}\frac{\partial x}{\partial r}\cos(\theta) + \frac{\partial^2 u}{\partial x \partial y}\frac{\partial y}{\partial r}\cos(\theta) + \frac{\partial^2 u}{\partial y^2}\frac{\partial y}{\partial r}\sin(\theta) + \frac{\partial^2 u}{\partial y \partial x}\frac{\partial x}{\partial r}\sin(\theta)$$

$$\frac{\partial^2 u}{\partial r^2} = \frac{\partial^2 u}{\partial x^2}\cos(\theta)^2 + 2\frac{\partial^2 u}{\partial x \partial y}\cos(\theta)\sin(\theta) + \frac{\partial^2 u}{\partial y^2}\sin(\theta)^2,$$

sowie

$$\frac{\partial u}{\partial \theta} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial \theta} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \theta}$$

$$\frac{\partial u}{\partial \theta} = -\frac{\partial u}{\partial x} r\sin(\theta) + \frac{\partial u}{\partial y} r\cos(\theta)$$

und

$$\frac{\partial^2 u}{\partial \theta^2} = -\frac{\partial^2 u}{\partial x^2}\frac{\partial x}{\partial \theta} r\sin(\theta) - \frac{\partial u}{\partial x}\frac{\partial}{\partial \theta} r\sin(\theta) - \frac{\partial^2 u}{\partial x \partial y}\frac{\partial y}{\partial \theta} r\sin(\theta) + \frac{\partial^2 u}{\partial y^2}\frac{\partial y}{\partial \theta} r\cos(\theta) + \frac{\partial u}{\partial y}\frac{\partial}{\partial \theta} r\cos(\theta) + \frac{\partial^2 u}{\partial y \partial x}\frac{\partial x}{\partial \theta} r\cos(\theta)$$

$$\frac{\partial^2 u}{\partial \theta^2} = \frac{\partial^2 u}{\partial x^2} r^2\sin(\theta)^2 - \frac{\partial u}{\partial x} r\cos(\theta) - \frac{\partial^2 u}{\partial x \partial y} r^2\sin(\theta)\cos(\theta) + \frac{\partial^2 u}{\partial y^2} r^2\cos(\theta)^2 - \frac{\partial u}{\partial y} r\sin(\theta) - \frac{\partial^2 u}{\partial x \partial y} r^2\sin(\theta)\cos(\theta)$$

Für $\frac{\partial^2 u}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}$ erhalten wir

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = \frac{\partial^2 u}{\partial x^2}(\cos(\theta)^2 + \sin(\theta)^2) + \frac{\partial^2 u}{\partial x \partial y}(2\sin(\theta)\cos(\theta) - 2\sin(\theta)\cos(\theta)) + \frac{\partial^2 u}{\partial y^2}(\sin(\theta)^2 + \cos(\theta)^2) - \frac{1}{r}(\frac{\partial u}{\partial x}\cos($$

Mit $\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x}\cos(\theta) + \frac{\partial u}{\partial y}\sin(\theta)$ und $\cos(\theta)^2 + \sin(\theta)^2 = 1$ erhalten wir

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{1}{r}\frac{\partial u}{\partial r}$$

und entsprechend für

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}$$

Dementsprechend gilt für $\Delta\phi$ mit $\Delta\phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}$ in Polarkoordinaten:

$$\Delta\phi(r, \epsilon) = \frac{\partial^2 \phi}{\partial r^2} + \frac{1}{r}\frac{\partial \phi}{\partial r} + \frac{1}{r^2}\frac{\partial^2 \phi}{\partial \theta^2}$$

Wobei ϕ hier nur von r abhängt, i.e. $\frac{\partial\phi}{\partial\theta} = 0$. Entsprechend gilt:

$$\Delta\phi(r,\epsilon) = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r}$$

Und damit:

$$\Delta\phi(r,\epsilon) = \frac{\epsilon^2((\epsilon r)^2 + 2)}{((\epsilon r)^2 + 1)^{\frac{3}{2}}}$$

Definiere ϕ (`phi`) und $\Delta\phi$ (`Lphi`) in `Python`:

[1]:
```
import numpy as np

def phi(r, e):
    return np.sqrt(1 + (e*r) ** 2)

def Lphi(r, e):
    return e ** 2 * ((e*r) ** 2 + 2) / ((e*r) ** 2 + 1) ** (3/2)
```

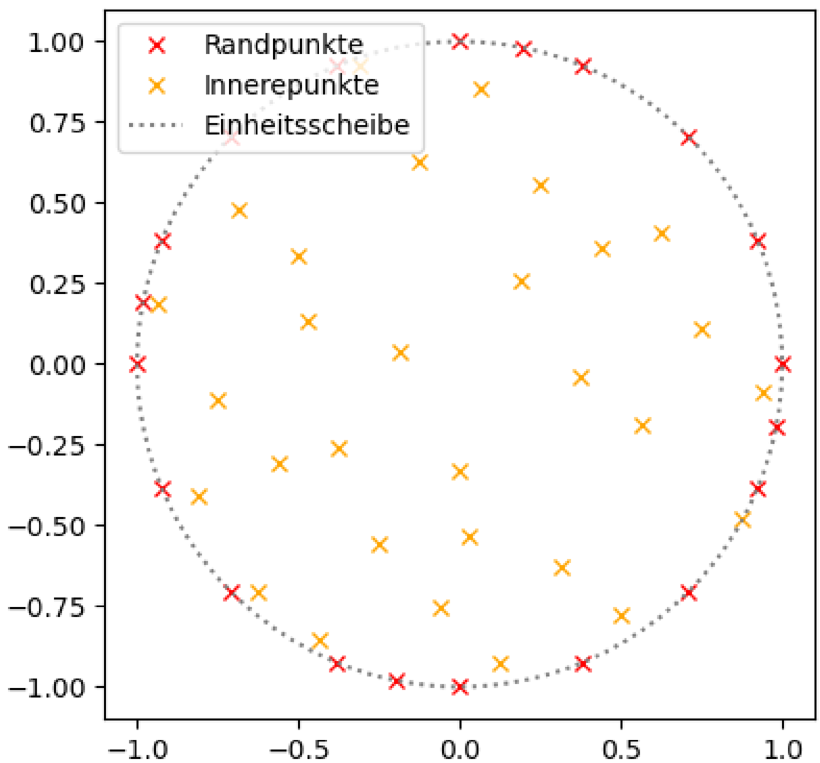
Definiere u_5 (`u`) und Δu_5 (`Lu`) in `Python`:

[2]:
```
def u(x, y):
    return np.sin(np.pi * (x ** 2 + y ** 2))

def Lu(x,y):
    return -4 * np.pi * (np.pi * (x ** 2 + y ** 2) * np.sin(np.pi * (x ** 2 + y␣
 ↪** 2)) - np.cos(np.pi * (x ** 2 + y ** 2)))
```

Erzeuge Distribution der 50 Nodes und Center-Points (entsprechend Methode 1 sind diese hier identisch) auf der Einheitsscheibe, wobei 30 im Inneren (`intd`) sind und 20 auf dem Rand (`bdyd`) liegen:

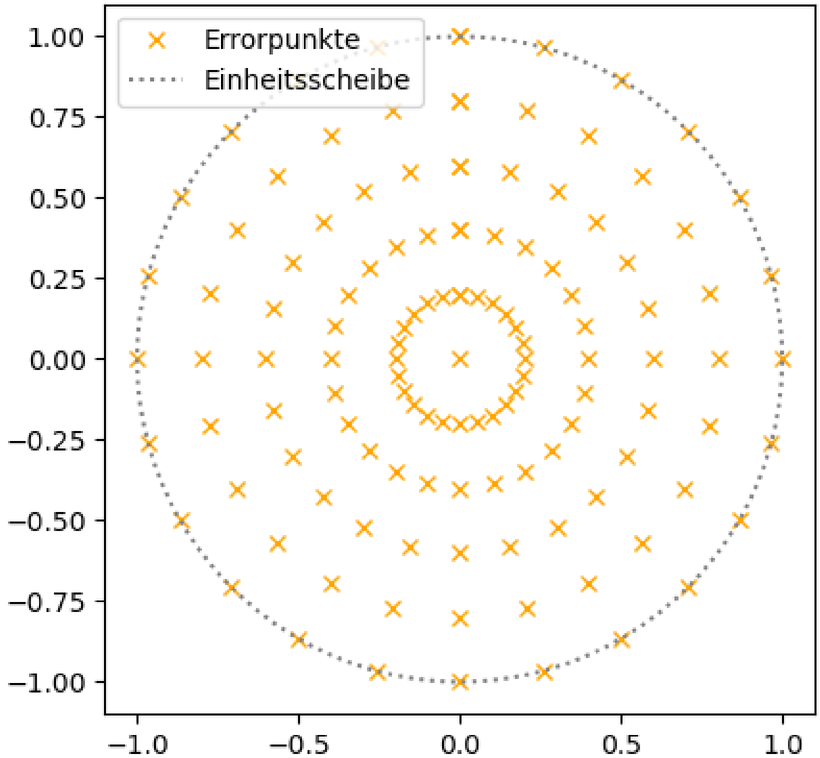
[3]:
```
from scipy.stats import qmc
import matplotlib.pyplot as plt

rng_bdy = qmc.Halton(d=1, scramble = False) # Pseudo-Zufallsgenerator für eine␣
 ↪Dimension (Winkel)
theta_bdy = rng_bdy.random(n = 20) * 2 * np.pi # Erzeuge 20 Zufallszahlen und␣
 ↪skaliere von [0, 1] auf [0, 2 * Pi]

# Mit diesen Zufallszahlen erzeuge Punkte auf Einheitskreis
bdyd_x = np.sin(theta_bdy)
bdyd_y = np.cos(theta_bdy)

bdyd = np.array([bdyd_x, bdyd_y]).T.reshape(20,2)

# Pseudo-Zufallsgenerator für zwei Dimension (x, y)
rng_int = qmc.Halton(d=2, scramble = False)

intd = [] # Array das am Ende die 30 Innerenpunkt (x, y) enthalten soll
```

```python
# Erzeuge "Monte-Carlo mäßig" 30 Punkte
while len(intd) < 30:
    intd_ = rng_int.random(n = 30 - len(intd)) * 2 - 1 # Erzeuge 30 - (Anzahl bereits erzeugter Werte) Zufallszahlen (x, y) auf [-1, 1]
    dist = np.linalg.norm(intd_, axis = 1) # Berechne Norm für diese Zufallszahlen (sqrt(x^2+y^2))
    ind = []
    for i, d in enumerate(dist): # Loop über alle erzeugten Zufallszahlen
        if d >= 1: # Wenn Norm der Zufallszahl >= 1 (i.e. Punkt liegt nicht in Einheitsscheibe), merke Index zum löschen
            ind.append(i)
    intd_ = np.delete(intd_, ind, axis = 0) # Lösche alle Punkte die außerhalb der Einheitsscheibe liegen aus Array
    if len(intd) == 0:
        intd = intd_
    else:
        intd = np.append(intd, intd_, axis = 0)

th = np.linspace(0, 2*np.pi, 100) # Äquidistante Winkeldistribution für das Ploten der Einheitsscheibe

#Plotten :)
plt.plot(bdyd_x, bdyd_y, color = 'red', linestyle = 'None', markersize = 5.5, marker = 'x', label = 'Randpunkte');
plt.plot(intd.T[0], intd.T[1], color = 'orange', linestyle = 'None', markersize = 5.5, marker = 'x', label = 'Innerepunkte')
plt.plot(np.sin(th), np.cos(th), color = 'gray', linestyle = 'dotted', label = 'Einheitsscheibe');
plt.gca().set_aspect('equal')
plt.legend(loc='upper left')
plt.show();
```



Erzeuge Punkte an denen der Error (`errd`) evaluiert wird:

[4]:
```
errd = np.array([[0, 0]])
theta = np.linspace(0, 2*np.pi, 25)
for i in range(5):
    x = np.sin(theta) * 1/5 * (i + 1)
    y = np.cos(theta) * 1/5 * (i + 1)
    d = [[x, y]]
    errd = np.append(errd, np.transpose(d).reshape((25, 2)), axis = 0)

th = np.linspace(0, 2*np.pi, 100) # Äquidistante Winkeldistribution für das␣
 ↪Ploten der Einheitsscheibe

#Plotten :)
plt.plot(errd.T[0], errd.T[1], color = 'orange', linestyle = 'None', markersize␣
 ↪= 5.5, marker = 'x', label = 'Errorpunkte')
plt.plot(np.sin(th), np.cos(th), color = 'gray', linestyle = 'dotted', label =␣
 ↪'Einheitsscheibe');
plt.gca().set_aspect('equal')
plt.legend(loc='upper left')
plt.show();
```



Der Error $E(\epsilon)$ ist definiert als

$$E(\epsilon) = \max_{\Omega} |s(x, \epsilon) - u(x)|$$

Definiere in `Python`:

[5]:
```python
def Err(s, u):
    return np.max(np.absolute(s - u))
```

Für das Problem

$$\Delta u(x) = f(x) \quad \text{in } \Omega$$
$$u(x) = g(x) \quad \text{on } \partial\Omega$$

Folgt mit $s(x, \epsilon) \approx u(x)$:

$$\Delta s(x, \epsilon) = f(x) \quad \text{in } \Omega$$
$$s(x, \epsilon) = g(x) \quad \text{on } \partial\Omega$$

Bzw. da in unserem Fall $f(x)$ und $g(x)$ aus einer vorgegebenen Lösung $u(x)$ definiert werden:

$$\Delta s(x, \epsilon) = \Delta u(x) \quad \text{in } \Omega$$

$$s(x, \epsilon) = u(x) \quad \text{on } \partial\Omega$$

Und entsprechend der Definition von $s(x, \epsilon)$:

$$\sum_{j=1}^{N_I} \lambda_j \Delta\phi(||x - x_j||, \epsilon) = \Delta u(x) \quad \text{in } \Omega$$

$$\sum_{j=1}^{N_B} \lambda_j \phi(||x - x_j||, \epsilon) = u(x) \quad \text{on } \partial\Omega$$

Mit

$$(A)_{ij} = \Delta\phi(||x_i - x_j||, \epsilon) \quad \text{für } j < N_I,$$
$$(A)_{ij} = \phi(||x_i - x_j||, \epsilon) \quad \text{für } N_I < j < N_I + N_B,$$
$$\lambda = (\lambda_1, ..., \lambda_{N_I}, \lambda_{N_I+1}, ..., \lambda_{N_I+N_B})$$

und

$$\tilde{u} = (\Delta u(x_i), u(x_j)) \quad \text{für } i < N_I, j < N_B$$

können wir das zusammenfassen zu:

$$A \cdot \lambda = \tilde{u}$$

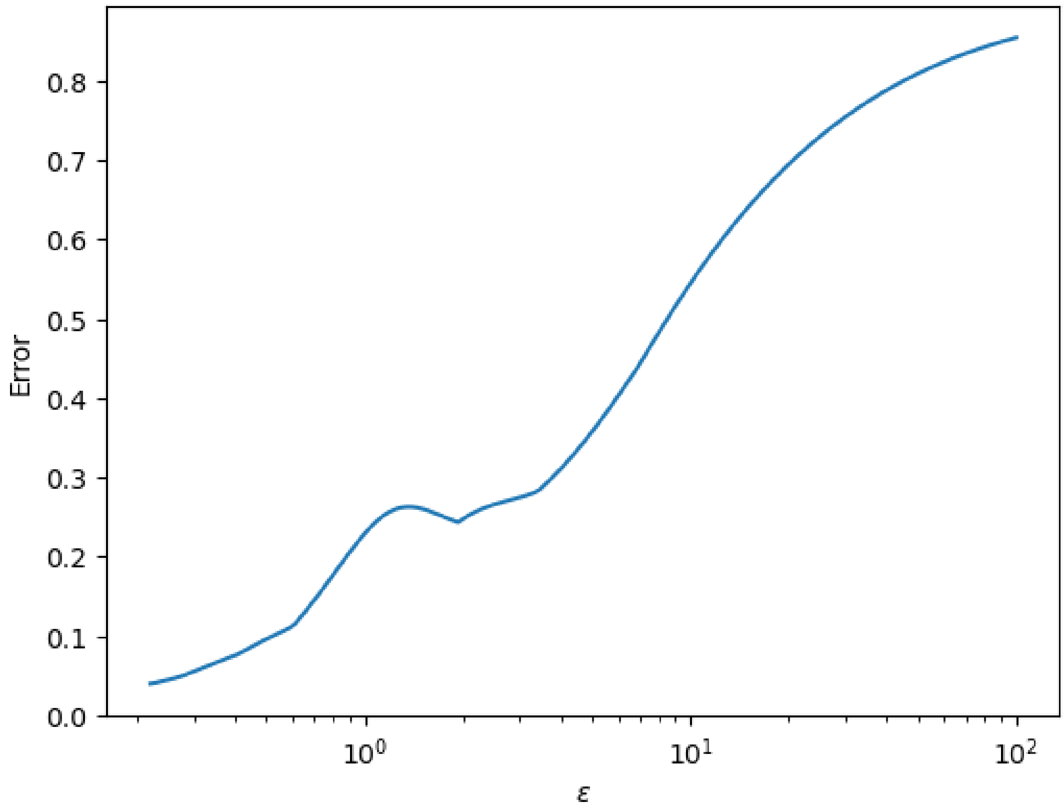
Dieses Gleichungssystem lässt sich lösen, da A (in den meisten Fällen) nicht singulär wird und wird hier `SciPy` überlassen:

[6]:
```
from scipy import linalg
from scipy.spatial import distance_matrix

def meth1_5(e, int, bdy, err):
    dm = distance_matrix(err, np.append(int, bdy, axis = 0))
    dm_int = distance_matrix(int, np.append(int, bdy, axis = 0))
    dm_bdy = distance_matrix(bdy, np.append(int, bdy, axis = 0))

    u_tilde = np.append(Lu(int.T[0], int.T[1]), u(bdy.T[0], bdy.T[1]), axis =
 ↪0).reshape(50,1)
    A = np.asarray(np.bmat([[Lphi(dm_int, e)], [phi(dm_bdy, e)]]))
    lam = linalg.solve(A, u_tilde)

    return np.dot(phi(dm, e), lam)
```

Auswertung des Fehlers:

[7]:
```
u_exact = u(errd.T[0], errd.T[1])
eps = np.geomspace(2.2e-1, 1e2, 150)
Es = []
for ep in eps:
    s_approx = meth1_5(ep, intd, bdyd, errd).flatten()
    Es.append(Err(s_approx, u_exact))

plt.plot(eps, Es);
plt.xlabel('$\epsilon$');
plt.ylabel('Error')
```

```
plt.xscale('log');
plt.show();
```



Für kleinere ϵ explodiert der Fehler und `SciPy` wirft Errors, da A nicht mehr well-posed ist.

1.3 Poisson Eq. Example (u_1 Larsson, Fornberg)

1.3.1 Approximation mit Methode 1 (Unsymmetrisch):

Um das Ergebnis nochmal mit dem des Papers vergleichen zu können betrachten wir u_1, da dieses geplottet wurde:

$$u_1 = \frac{65}{65 + (x - 0.2)^2 + (y + 0.1)^2}$$

Berechnen von Δu_1 mit `SymPy`:

$$\Delta u_1 =$$

[8]:
```
from sympy import *

def u1(x, y):
    return 65 / (65 + (x - 0.2)**2 + (y + 0.1) ** 2)
```

```
x, y = symbols('x y')
init_session(quiet = True)
u1_ = 65 / (65 + (x - 0.2)**2 + (y + 0.1) ** 2)
diff(u1_, x, x) + diff(u1_, y, y)
```

[8]:

$$\frac{65\left(\frac{(2x-0.4)(4x-0.8)}{(x-0.2)^2+(y+0.1)^2+65}-2\right)}{\left((x-0.2)^2+(y+0.1)^2+65\right)^2}+\frac{65\left(\frac{(2y+0.2)(4y+0.4)}{(x-0.2)^2+(y+0.1)^2+65}-2\right)}{\left((x-0.2)^2+(y+0.1)^2+65\right)^2}$$

[9]:

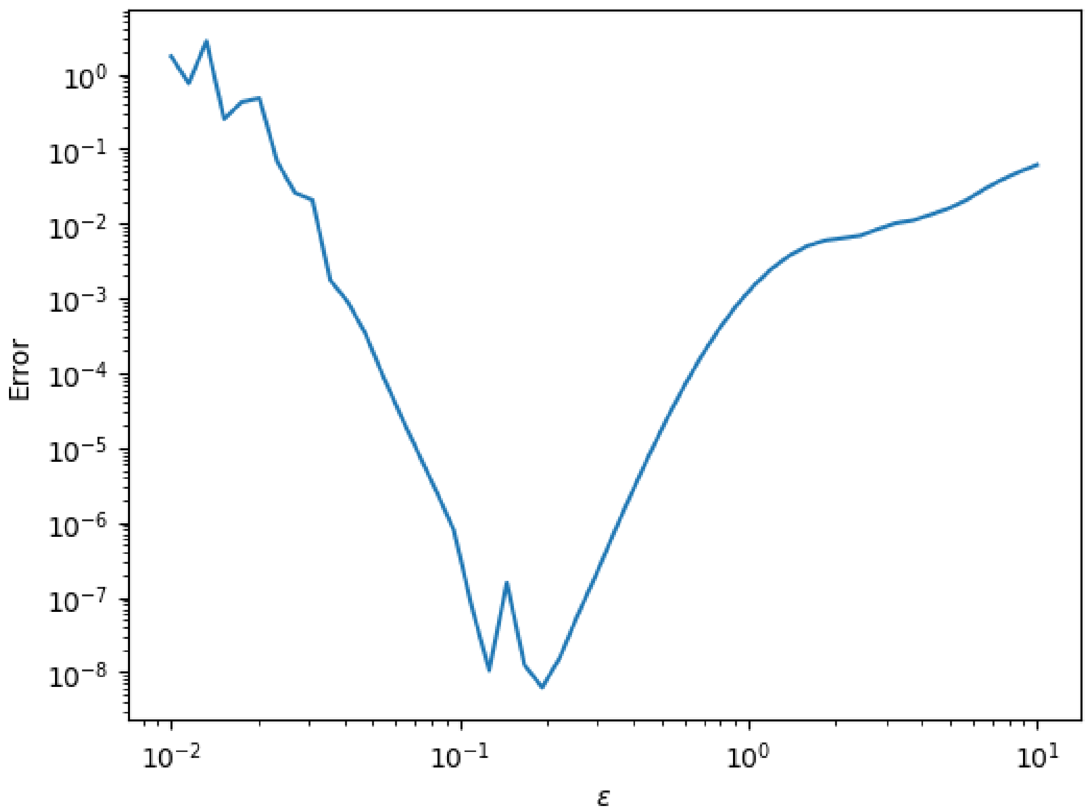
```
Lu1 = lambdify((x, y), diff(u1_, x, x) + diff(u1_, y, y))
```

[10]:

```
def meth1_1(e, int, bdy, err):
    dm = distance_matrix(err, np.append(int, bdy, axis = 0))
    dm_int = distance_matrix(int, np.append(int, bdy, axis = 0))
    dm_bdy = distance_matrix(bdy, np.append(int, bdy, axis = 0))

    u_tilde = np.append(Lu1(int.T[0], int.T[1]), u1(bdy.T[0], bdy.T[1]), axis =␣
 ↪0).reshape(50,1)
    A = np.asarray(np.bmat([[Lphi(dm_int, e)], [phi(dm_bdy, e)]]))
    lam = linalg.solve(A, u_tilde)

    return np.dot(phi(dm, e), lam)
```

[11]:

```
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

u_exact = u1(errd.T[0], errd.T[1])
eps = np.geomspace(1e-2, 1e1, 50)
Es = []
for ep in eps:
    s_approx = meth1_1(ep, intd, bdyd, errd).flatten()
    Es.append(Err(s_approx, u_exact))

plt.plot(eps, Es);
plt.xlabel('$\epsilon$');
plt.ylabel('Error')
plt.xscale('log');
plt.yscale('log');
plt.show();
```



Das schaut sehr wie das Ergebnis im Paper aus. Auch hier für kleinere ϵ ist A nicht mehr well-posed.

1.3.2 Approximation mit Methode 2 (Symmetrisch):

Um definitiv zu verhindern, dass A singulär wird, können wir $s(x,\epsilon)$ so definieren, dass A symmetrisch ist:

$$s(x,\epsilon) = \sum_{j=1}^{N_B} \lambda_j \phi(||x - x_j||, \epsilon) + \sum_{j=N_B+1}^{N} \lambda_j \Delta\phi(||x - x_j||, \epsilon)$$

Durch einsetzen in die PDE gilt:

$$\sum_{j=1}^{N_B} \lambda_j \phi(||x_i - x_j||, \epsilon) + \sum_{j=N_B+1}^{N} \lambda_j \Delta\phi(||x_i - x_j||, \epsilon) = g(x_i)$$

Für $i = 1, ..., N_B$.
Und

$$\sum_{j=1}^{N_B} \lambda_j \Delta\phi(||x_i - x_j||, \epsilon) + \sum_{j=N_B+1}^{N} \lambda_j \Delta^2\phi(||x_i - x_j||, \epsilon) = f(x_i)$$

Für $i = N_B + 1, ..., N$.
Für die hier genutzte `RBF` gilt (mit `SymPy`)

$$\Delta^2\phi(r,\epsilon) =$$

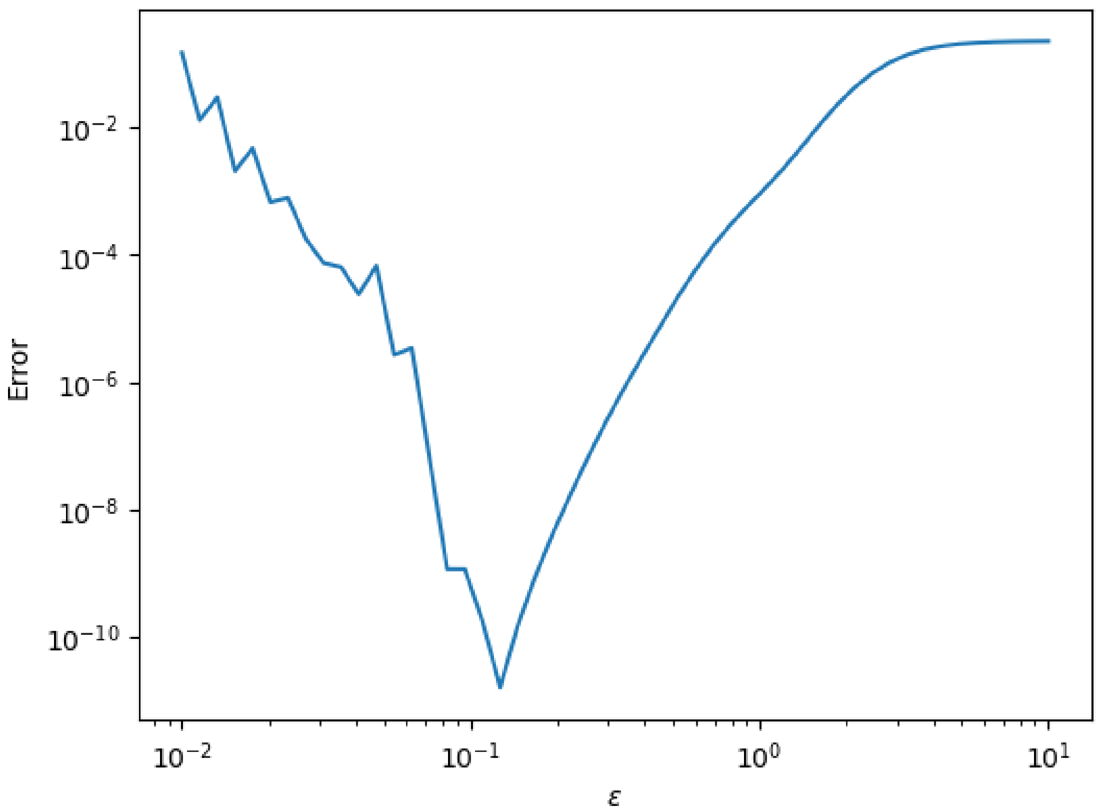
[12]:
```python
r, e = symbols('r e')
phi__ = e ** 2 * ((e*r) ** 2 + 2) / ((e*r) ** 2 + 1) ** (3/2)

simplify(simplify(diff(phi__, r, r)+1/r*diff(phi__, r)))
```

[12]:
$$\frac{e^4\left(e^2r^2\left(e^2r^2+1\right)^{4.0}\cdot\left(15.0e^2r^2+30.0\right)-\left(e^2r^2+1\right)^{5.0}\cdot\left(18.0e^2r^2+12.0\right)+4.0\left(e^2r^2+1\right)^{6.0}\right)}{\left(e^2r^2+1\right)^{7.5}}$$

[13]:
```python
L2phi = lambdify((r, e), simplify(simplify(diff(phi__, r, r)+1/r*diff(phi__,␣
↪r))))
```

[14]:
```python
def meth2_1(e, int, bdy, err):
    dm_int = distance_matrix(err, int)
    dm_bdy = distance_matrix(err, bdy)
    dm_int_int = distance_matrix(int, int)
    dm_int_bdy = distance_matrix(int, bdy)
    dm_bdy_int = distance_matrix(bdy, int)
    dm_bdy_bdy = distance_matrix(bdy, bdy)

    u_tilde = np.append(Lu1(int.T[0], int.T[1]), u1(bdy.T[0], bdy.T[1]), axis =␣
↪0).reshape(50,1)
    A = np.asarray(np.bmat([[Lphi(dm_int_bdy, e), L2phi(dm_int_int, e)],␣
↪[phi(dm_bdy_bdy, e), Lphi(dm_bdy_int, e)]]))
    lam = linalg.solve(A, u_tilde)
    return np.dot(np.asarray(np.bmat([phi(dm_bdy, e), Lphi(dm_int, e)])), lam)
```

[15]:
```python
u_exact = u1(errd.T[0], errd.T[1])
eps = np.geomspace(1e-2, 1e1, 50)
Es = []
for ep in eps:
    s_approx = meth2_1(ep, intd, bdyd, errd).flatten()
    Es.append(Err(s_approx, u_exact))

plt.plot(eps, Es);
plt.xlabel('$\epsilon$');
plt.ylabel('Error')
plt.xscale('log');
plt.yscale('log');
plt.show();
```



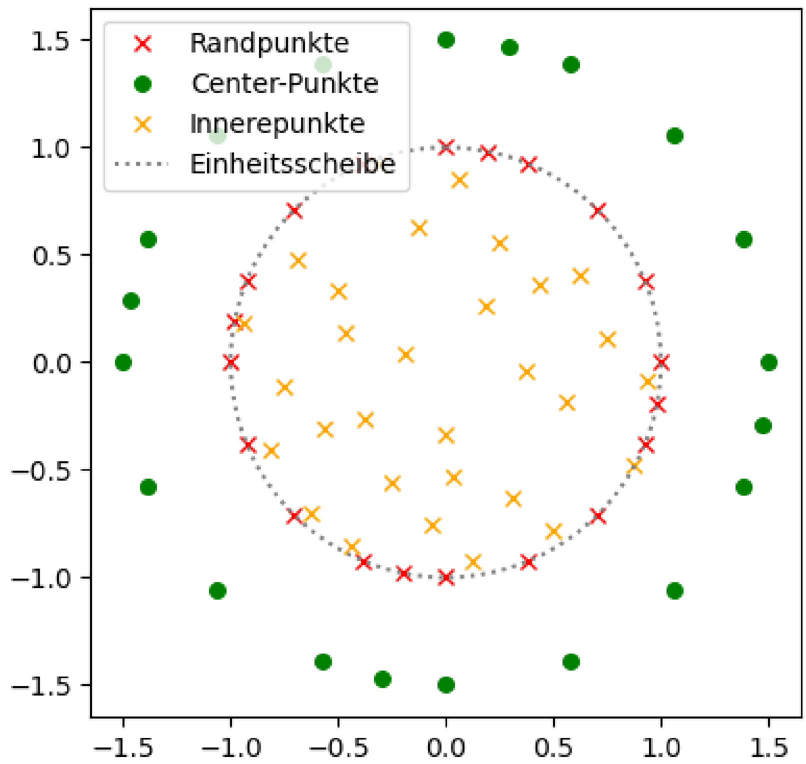
Vergleiche mit Methode 1, Methode 2 ist für kleine ϵ genauer.

1.3.3 Approximation mit Methode 3 (Unsymmetrisch):

(Beschreibung noch schreiben)

[16]:
```
rng_cnt = qmc.Halton(d=1, scramble = False) # Pseudo-Zufallsgenerator für eine␣
↪Dimension (Winkel)
theta_cnt = rng_bdy.random(n = 20) * 2 * np.pi # Erzeuge 20 Zufallszahlen und␣
↪skaliere von [0, 1] auf [0, 2 * Pi]

# Mit diesen Zufallszahlen erzeuge Punkte auf Kreis mit Radius 1.5
cntd_x = 1.5 * np.sin(theta_bdy)
cntd_y = 1.5 * np.cos(theta_bdy)

cntd = np.array([cntd_x, cntd_y]).T.reshape(20,2)

th = np.linspace(0, 2*np.pi, 100) # Äquidistante Winkeldistribution für das␣
↪Ploten der Einheitsscheibe

#Plotten :)
```

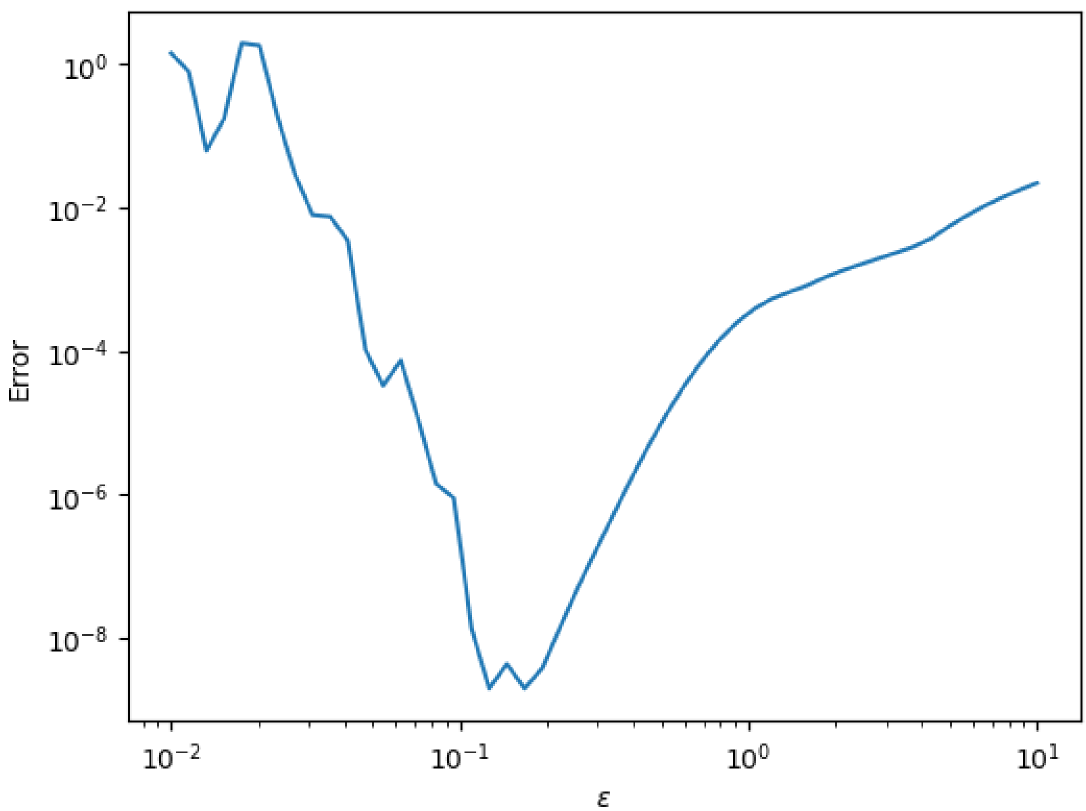
```
plt.plot(bdyd_x, bdyd_y, color = 'red', linestyle = 'None', markersize = 5.5,␣
 ↪marker = 'x', label = 'Randpunkte');
plt.plot(cntd_x, cntd_y, color = 'green', linestyle = 'None', markersize = 5.5,␣
 ↪marker = 'o', label = 'Center-Punkte');
plt.plot(intd.T[0], intd.T[1], color = 'orange', linestyle = 'None', markersize␣
 ↪= 5.5, marker = 'x', label = 'Innerepunkte')
plt.plot(np.sin(th), np.cos(th), color = 'gray', linestyle = 'dotted', label =␣
 ↪'Einheitsscheibe');
plt.gca().set_aspect('equal')
plt.legend(loc='upper left')
plt.show();
```



[17]:
```
def meth3_1(e, int, bdy, cnt, err):
    dm = distance_matrix(err, np.append(int, cnt, axis = 0))
    dm_int = distance_matrix(int, np.append(int, cnt, axis = 0))
    dm_bdy = distance_matrix(bdy, np.append(int, cnt, axis = 0))

    u_tilde = np.append(Lu1(int.T[0], int.T[1]), u1(bdy.T[0], bdy.T[1]), axis =␣
 ↪0).reshape(50,1)
    A = np.asarray(np.bmat([[Lphi(dm_int, e)], [phi(dm_bdy, e)]]))
    lam = linalg.solve(A, u_tilde)
```

```
    return np.dot(phi(dm, e), lam)
```

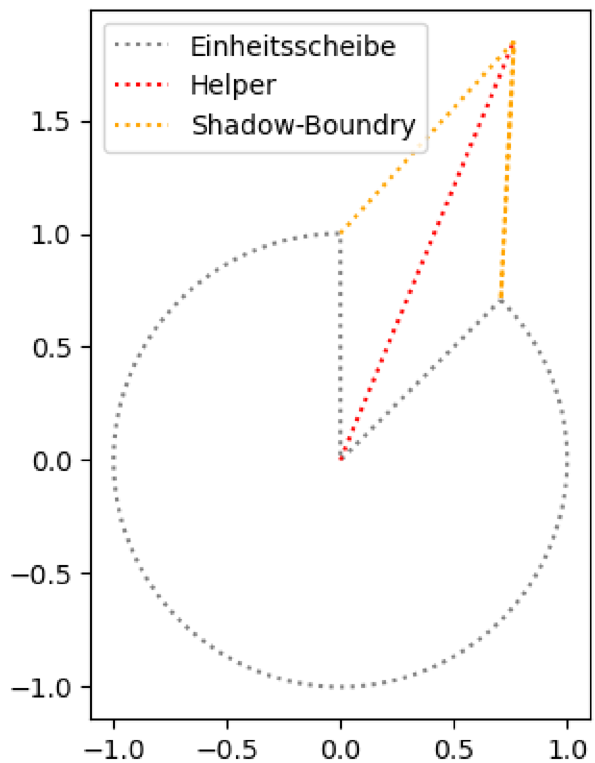
[18]:
```
u_exact = u1(errd.T[0], errd.T[1])
eps = np.geomspace(1e-2, 1e1, 50)
Es = []
for ep in eps:
    s_approx = meth3_1(ep, intd, bdyd, cntd, errd).flatten()
    Es.append(Err(s_approx, u_exact))

plt.plot(eps, Es);
plt.xlabel('$\epsilon$');
plt.ylabel('Error')
plt.xscale('log');
plt.yscale('log');
plt.show();
```



1.3.4 Vergleich der Methoden für "kantiges" Ω, Beispiel 1

Wähle Ω als Einheitskreisscheibe mit rausgeschnittenem "Kuchenstück"

[19]:
```python
N_B_o = 1000

th = np.linspace(1/4*np.pi, 2*np.pi, N_B_o) # Äquidistante Winkeldistribution␣
↪für das Ploten der Einheitsscheibe
bdy = np.array([[np.sin(th)], [np.cos(th)]]).reshape(2, N_B_o)
n_dx = np.floor(1 / (7/4 * np.pi / N_B_o)).astype(int)
bdy2 = np.array([np.zeros((n_dx, 1)), np.linspace(0, 1, n_dx).reshape(n_dx,1)]).
↪reshape(2,n_dx)
bdy3 = np.array([np.linspace(0, np.sin(1/4 * np.pi), n_dx).reshape(n_dx,1), np.
↪linspace(0, np.cos(1/4 * np.pi), n_dx).reshape(n_dx,1)]).reshape(2,n_dx)

bdy = np.append(bdy, bdy2, axis = 1)
bdy = np.append(bdy, bdy3, axis = 1)

help = np.array([np.linspace(0, 2 * np.sin(1/8 * np.pi), n_dx).reshape(n_dx,1),␣
↪np.linspace(0, 2 * np.cos(1/8 * np.pi), n_dx).reshape(n_dx,1)]).
↪reshape(2,n_dx)
help2 = np.array([np.linspace(0, 2 * np.sin(1/8 * np.pi), n_dx).
↪reshape(n_dx,1), np.linspace(1, 2 * np.cos(1/8 * np.pi), n_dx).
↪reshape(n_dx,1)]).reshape(2,n_dx)
help3 = np.array([np.linspace(np.sin(1/4 * np.pi), 2 * np.sin(1/8 * np.pi),␣
↪n_dx).reshape(n_dx,1), np.linspace(np.cos(1/4 * np.pi), 2 * np.cos(1/8 * np.
↪pi), n_dx).reshape(n_dx,1)]).reshape(2,n_dx)

help2 = np.append(help2, help3, axis = 1)

#Plotten :)
plt.plot(bdy[0], bdy[1], color = 'gray', linestyle = 'dotted', label =␣
↪'Einheitsscheibe');
plt.plot(help[0], help[1], color = 'red', linestyle = 'dotted', label =␣
↪'Helper');
plt.plot(help2[0], help2[1], color = 'orange', linestyle = 'dotted', label =␣
↪'Shadow-Boundry');
plt.gca().set_aspect('equal')
plt.legend(loc='upper left')
plt.show();
```




```
[ ]: rng_bdy = qmc.Halton(d=1, scramble = False) # Pseudo-Zufallsgenerator für eine␣
      ↪Dimension (Winkel)
     theta_bdy = rng_bdy.random(n = 20) * 2 * np.pi # Erzeuge 20 Zufallszahlen und␣
      ↪skaliere von [0, 1] auf [0, 2 * Pi]

     # Mit diesen Zufallszahlen erzeuge Punkte auf Einheitskreis

     bdyd2_x = []
     bdyd2_y = []

     for t in theta_bdy:
         if t > 1/4 * np.pi:
             bdyd2_x.append(np.sin(t))
             bdyd2_y.append(np.cos(t))
         else if t > 1/8 * np.pi:
             # Hier weiter schreiben!!!!!!!!

     bdyd = np.array([bdyd_x, bdyd_y]).T.reshape(20,2)

     # Pseudo-Zufallsgenerator für zwei Dimension (x, y)
     rng_int = qmc.Halton(d=2, scramble = False)
```

```python
intd = [] # Array das am Ende die 30 Innerenpunkt (x, y) enthalten soll

# Erzeuge "Monte-Carlo mäßig" 30 Punkte
while len(intd) < 30:
    intd_ = rng_int.random(n = 30 - len(intd)) * 2 - 1 # Erzeuge 30 - (Anzahl bereits erzeugter Werte) Zufallszahlen (x, y) auf [-1, 1]
    dist = np.linalg.norm(intd_, axis = 1) # Berechne Norm für diese Zufallszahlen (sqrt(x^2+y^2))
    ind = []
    for i, d in enumerate(dist): # Loop über alle erzeugten Zufallszahlen
        if d >= 1: # Wenn Norm der Zufallszahl >= 1 (i.e. Punkt liegt nicht in Einheitsscheibe), merke Index zum löschen
            ind.append(i)
    intd_ = np.delete(intd_, ind, axis = 0) # Lösche alle Punkte die außerhalb der Einheitsscheibe liegen aus Array
    if len(intd) == 0:
        intd = intd_
    else:
        intd = np.append(intd, intd_, axis = 0)

th = np.linspace(0, 2*np.pi, 100) # Äquidistante Winkeldistribution für das Ploten der Einheitsscheibe

#Plotten :)
plt.plot(bdyd_x, bdyd_y, color = 'red', linestyle = 'None', markersize = 5.5, marker = 'x', label = 'Randpunkte');
plt.plot(intd.T[0], intd.T[1], color = 'orange', linestyle = 'None', markersize = 5.5, marker = 'x', label = 'Innerepunkte')
plt.plot(np.sin(th), np.cos(th), color = 'gray', linestyle = 'dotted', label = 'Einheitsscheibe');
plt.gca().set_aspect('equal')
plt.legend(loc='upper left')
plt.show();
```