Table of Contents

Matplotlib (Teil 2)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (9., 3.)
plt.rcParams['lines.linewidth'] = 3
#%matplotlib qt
%matplotlib notebook
```

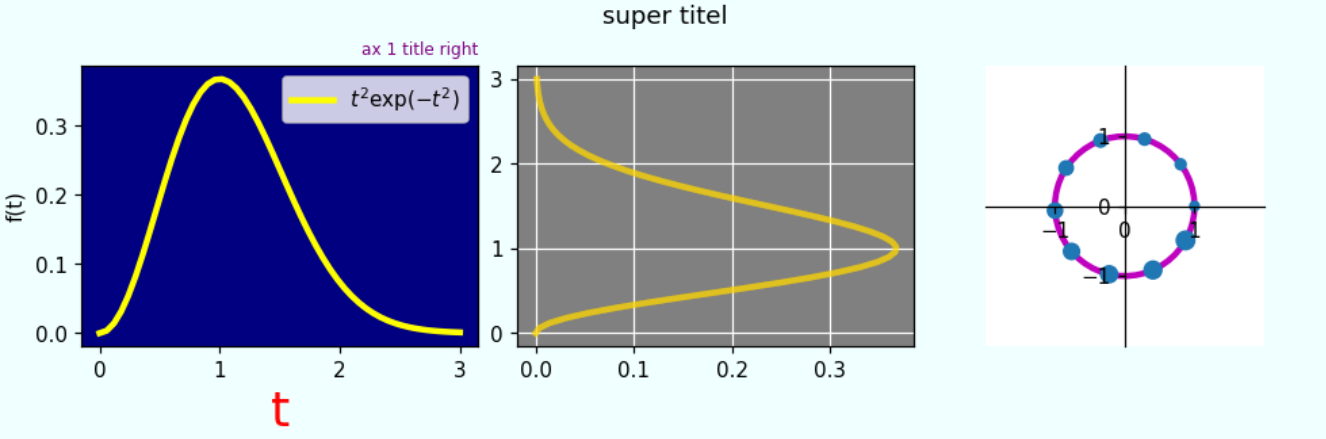
In [2]:
```
#plt.rcParams.keys()
```

https://matplotlib.org/stable/gallery/color/named_colors.html

Wiederholung

In [3]:
```
fig, ax = plt.subplots(1, 3, layout='constrained')
fig.suptitle('super titel')
fig.set_facecolor('azure')

ax[0].set_title('ax 1 title right', loc='right', color='purple', fontsize=8)
ax[0].set_facecolor('navy')
```



In [4]:
```
def f(t):
    return t**2 * np.exp(-t**2)


t = np.linspace(0, 3, 51)
y = f(t)

ax[0].plot(t, y, color='yellow', label='$t^2\\exp(-t^2)$')
ax[0].legend();
```

Out[4]:
```
<matplotlib.legend.Legend at 0x7f3301950da0>
```

Achsen und Gitterlinien

Hinzufügen von Achsenbezeichnungen. Das *layout='constrained'* in der subplots methode sorgt dafür das alles sichtbar bleibt.

In [7]:
```
ax[0].set_xlabel('t', color='red', fontsize=24)
ax[0].set_ylabel('f(t)');
```

In [8]:
```
ax[1].set_facecolor('grey')
ax[1].grid(color='white')
ax[1].plot(y, t, color='gold', alpha=.5);
```

In [9]:
```
phi = np.linspace(0, 2 * np.pi)
ax[2].clear()
ax[2].plot(np.cos(phi), np.sin(phi), 'm', zorder=1)

ax[2].scatter(np.cos(phi[::5]), np.sin(phi[::5]), s=20 + 10 * phi[::5])

ax[2].set_aspect('equal')

# Setze die bottom und left Achsen (engl. spines) als x und y Achse
ax[2].spines['bottom'].set_position('zero')
ax[2].spines['left'].set_position('zero')
# Entferne top und right spines
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)
```

In [11]:
```
ax[2].set_xlim((-2, 2))
ax[2].set_ylim((-2, 2))
ax[2].set_xticks([-1, 0, 1])
ax[2].set_yticks([-1, 0, 1]);
```

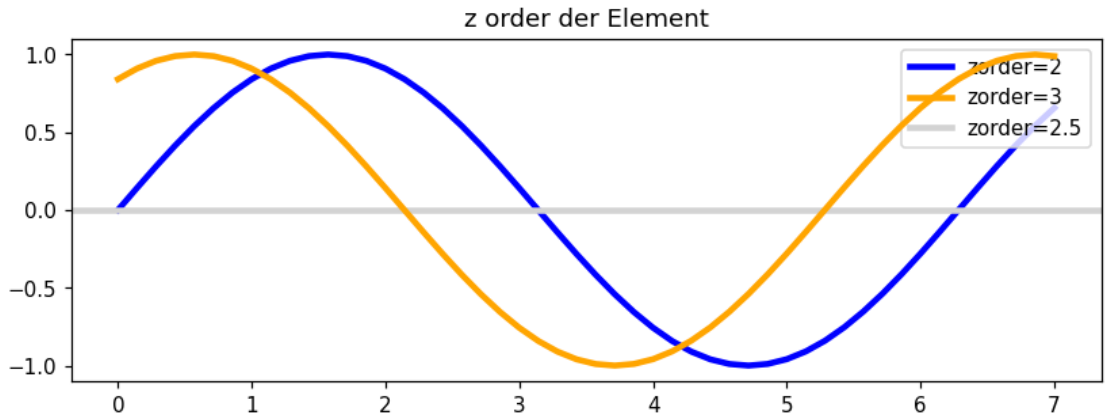
Reihenfolge der Elemente festlegen

zorder legt fest in welcher Reihenfolge die Elemente gezeichnet werden.

In [12]:
```
x = np.linspace(0, 7)

plt.figure()
plt.plot(x, np.sin(x), label='zorder=2', color='blue', zorder=2)
plt.plot(x, np.sin(x + 1), label='zorder=3', color='orange', zorder=3)
plt.axhline(0, label='zorder=2.5', color='lightgrey', zorder=2.5)

plt.title('z order der Element')
l = plt.legend(loc='upper right')
l.set_zorder(2.5)
plt.show()
```



Position der Legende

Location String	Location Code
'best'	0

'upper right' | 1 'upper left' | 2 'lower left' | 3 'lower right' | 4 'right' | 5 'center left' | 6 'center right'| 7 'lower center'| 8 'upper center'| 9 'center' | 10

Bilder abspeichern

Bild in einer Datei (pdf, jpeg, ...) speichern:

savefig

In [13]:
```
fig.savefig('bild1.pdf')  # das format wird anhand der Endung geraten
```

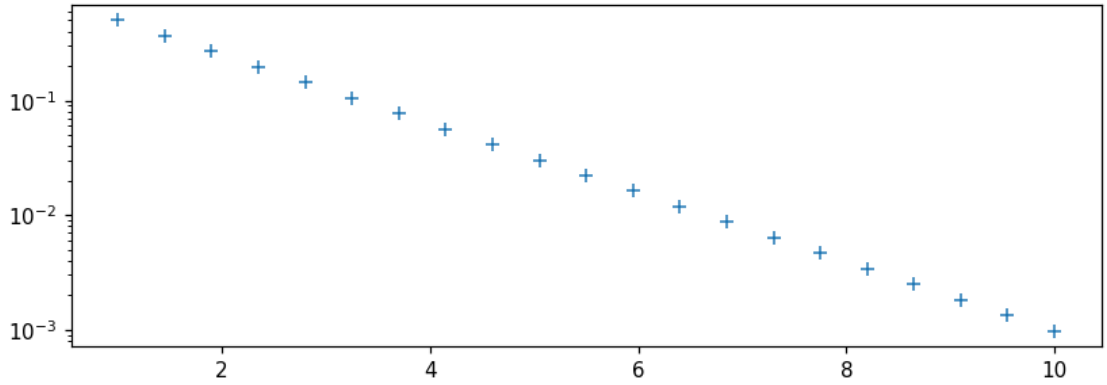
In [14]:
```
fig.savefig('bild1.jpg', format='jpg')
```

In [16]:
```
#fig.canvas.get_supported_filetypes_grouped()
```

Logarithmische Skalierung

Verwendung logarithmischer Achsen:

In [18]:
```
x = np.linspace(1, 10, 21)
y = 2**(-x)
fig, ax = plt.subplots(1)
ax.semilogy(x, y, '+');
```



Siehe auch semilogx() und loglog(), wenn die x-Achse bzw. beide Achsen logarithmisch skaliert

werden sollen

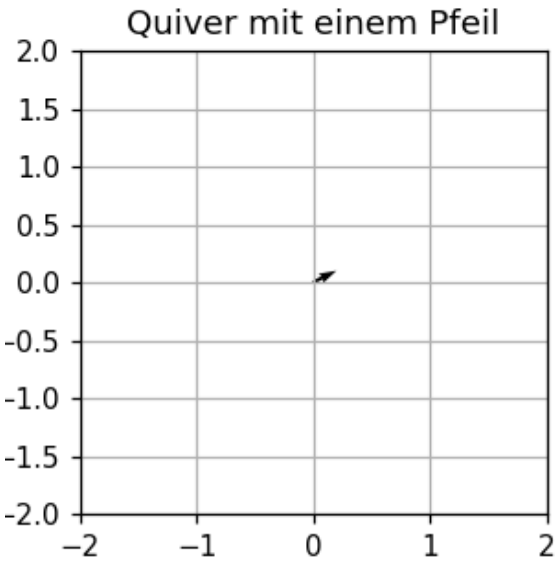
Vektorpfeile

Vektorpfeile mit

quiver

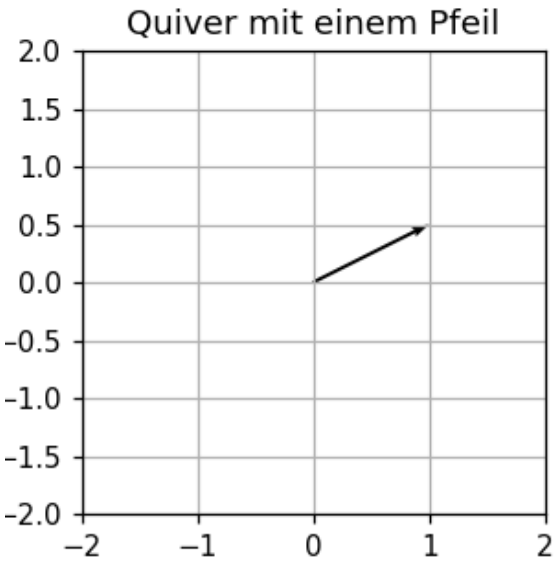
In [19]:
```
fig, ax = plt.subplots(figsize=(3, 3))
x, y, u, v = 0, 0, 1, .5

ax.quiver(x, y, u, v)  # (x,y) Fusspunkt (u,v) Richtung
ax.set_title('Quiver mit einem Pfeil')
ax.axis([-2, 2, -2, 2])
ax.grid()
```



In [20]:
```
fig, ax = plt.subplots(figsize=(3, 3))
x, y, u, v = 0, 0, 1, .5

ax.quiver(x, y, u, v, angles='xy', scale_units='xy', scale=1)
ax.set_title('Quiver mit einem Pfeil')
ax.axis([-2, 2, -2, 2])
ax.grid()
ax.set_aspect('equal')
```

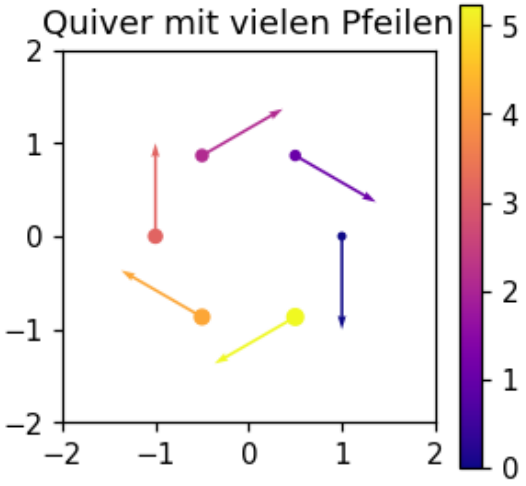


In [21]:
```
fig, ax = plt.subplots(figsize=(3, 3))
```

```
n = 7
t = np.linspace(0, 2 * np.pi, n)[:-1]
x = np.cos(t)
y = np.sin(t)
u = np.sin(t)
v = -np.cos(t)
color = t
ax.scatter(x, y, s=5 * t + 5, c=color, cmap='plasma')

qv = ax.quiver(x, y, u, v,  color, angles='xy', scale_units='xy',
               scale=1, cmap='plasma')
ax.set_title('Quiver mit vielen Pfeilen')
ax.axis([-2, 2, -2, 2])
ax.set_aspect('equal')

fig.colorbar(qv);
```

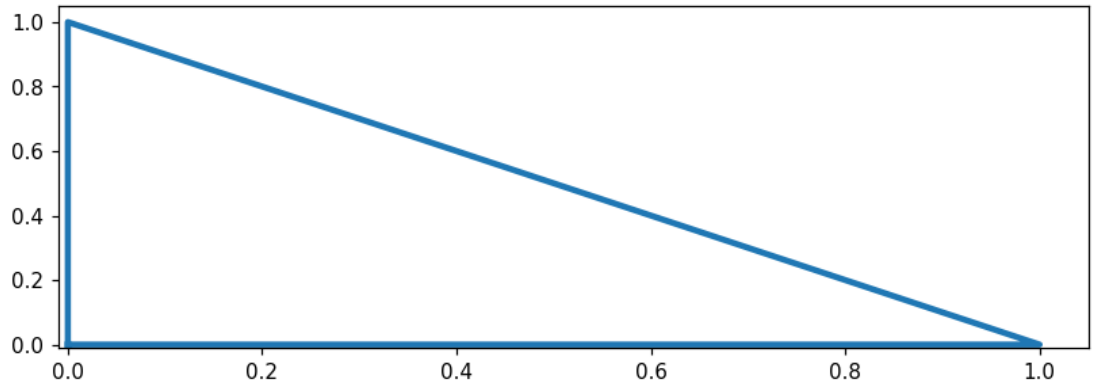


Der Parameter **cmap** wählt die Colormap aus.
Link zu Colormaps (cmap) https://matplotlib.org/stable/tutorials/colors/colormaps.html

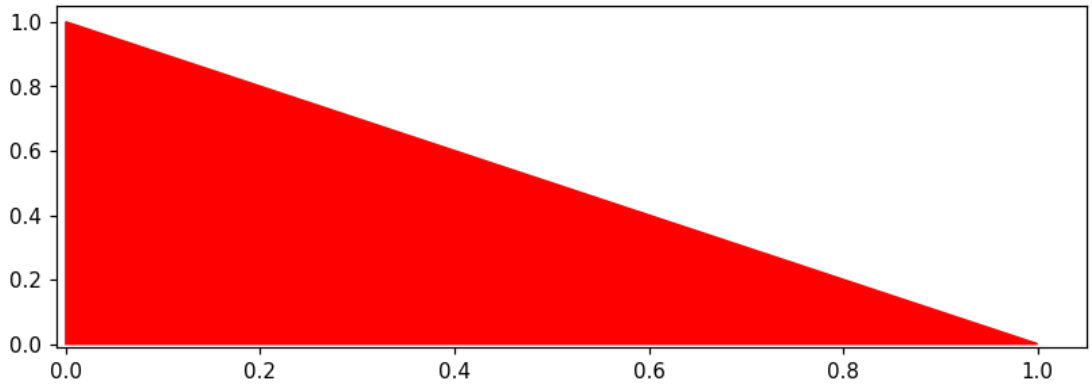
Flächen füllen

fill

In [23]:
```
fig, ax = plt.subplots(1)
ax.plot([0, 1, 0, 0], [0, 0, 1, 0])
ax.axis(xmin=-.01, ymin=-.01);
```

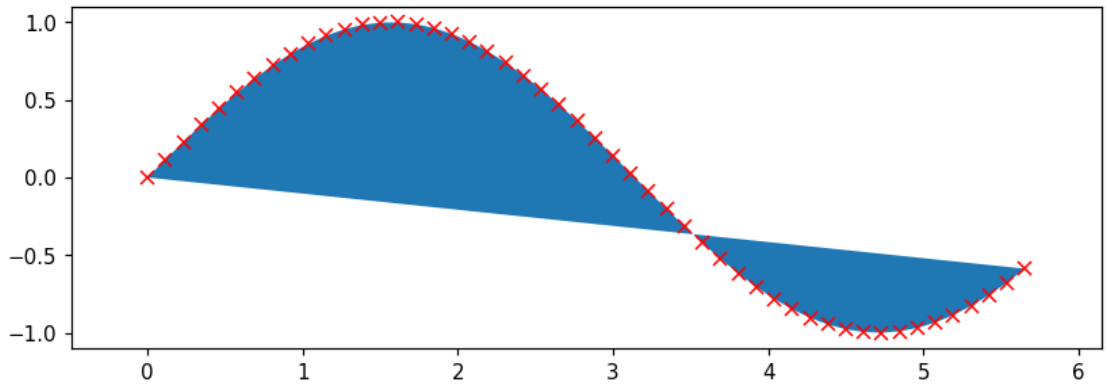


In [24]:
```
fig, ax = plt.subplots(1)
ax.fill([0, 1, 0, 0], [0, 0, 1, 0], color='red')
ax.axis(xmin=-.01, ymin=-.01);
```



In [25]:
```
x = np.linspace(0, 1.8 * np.pi)
y1 = np.sin(x)
```

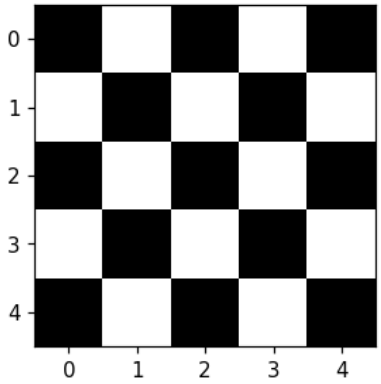
In [27]:
```
fig, ax = plt.subplots(1)
ax.fill(x, y1)
ax.plot(x, y1, 'xr')
ax.axis('equal');
```



Zeichnen von 2d reellwertigen Daten

imshow

In [29]:
```
M = (-1)**np.arange(25).reshape(5, 5)
fig, ax = plt.subplots(1)
ax.imshow(M, cmap='Greys');
```



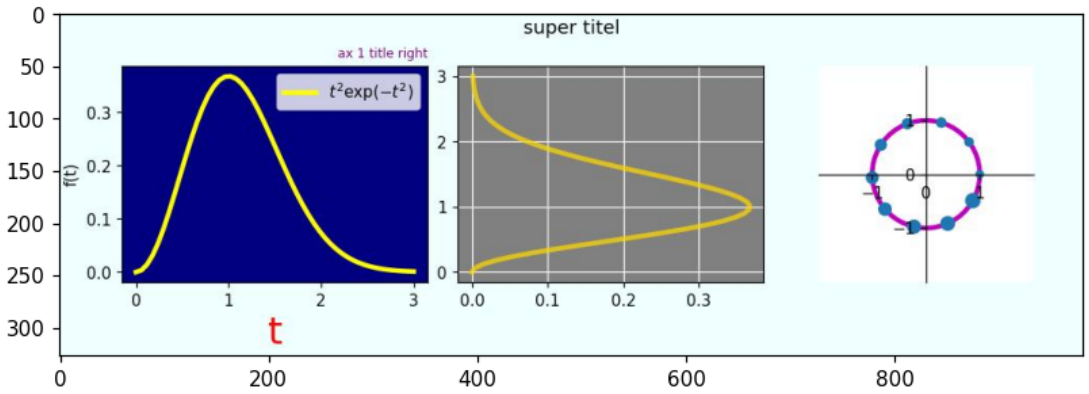
In [ ]:
```
plt.close('all')
```

In [30]:
```
from matplotlib.image import imread
```

```
img = imread('bild1.jpg')
img.shape
```

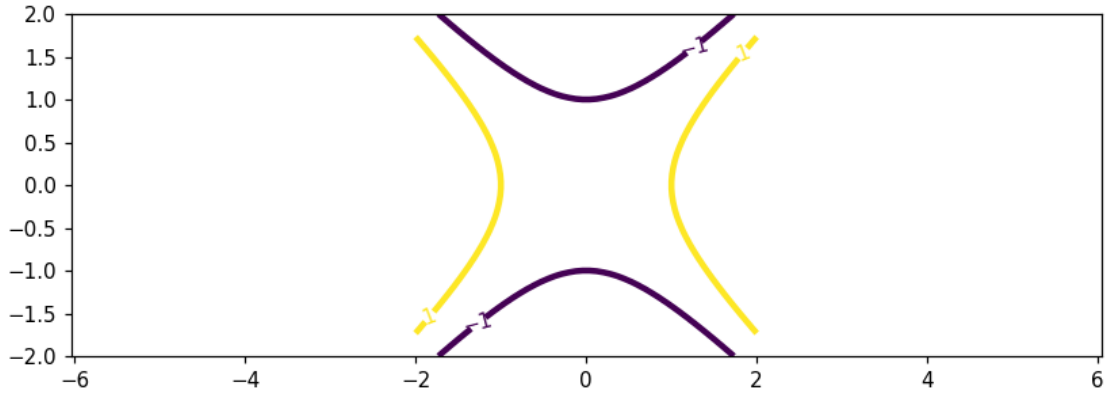
Out[30]: (327, 981, 3)

In [31]:
```
plt.figure()
plt.imshow(img)
```



Out[31]: <matplotlib.image.AxesImage at 0x7f32f6110500>

In [33]:
```
fig, ax = plt.subplots(1)
x = np.linspace(-2, 2)
y = np.linspace(-2, 2)
xx, yy = np.meshgrid(x, y)
zz = xx**2 - yy**2
cs = ax.contour(xx, yy, zz, [-1, 1])
cs.clabel()
ax.axis('equal');
```

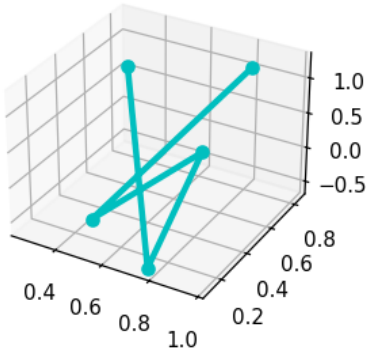


3D Plots

Polygonzug in 3d mit **plot3D**

In [35]:
```
x = np.random.rand(5)
y = np.random.rand(5)
z = np.random.randn(5)

ax = plt.figure().add_subplot(projection='3d')
ax.plot(x, y, z, 'co-');
```



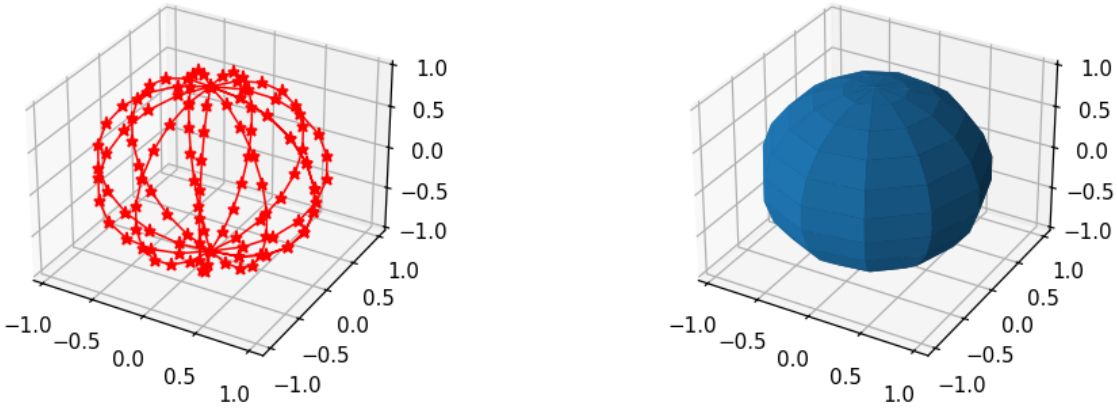
Fläche in 3d mit **surface**

In [38]:
```
fig, ax = plt.subplots(1, 2, layout='constrained', subplot_kw={'projection': '3d'})

phi = np.linspace(0, 2 * np.pi, 12)
phi = np.append(phi, np.nan).reshape(
    -1, 1)  # NaN Trick damit keine ungewollten Linien entstehen
theta = np.linspace(0, np.pi, 12)
theta = np.append(theta, np.nan).reshape(1, -1)

# Parametrisierung der Kugelfläche siehe https://de.wikipedia.org/wiki/Kugel
x = np.cos(phi) * np.sin(theta)  # durch broadcasting sind x, y, z hier 2D arrays
y = np.sin(phi) * np.sin(theta)
z = np.ones_like(phi) * np.cos(theta)

ax[0].plot(x.flatten(), y.flatten(), z.flatten(), 'r*-', linewidth=1)
ax[1].plot_surface(x, y, z);
```

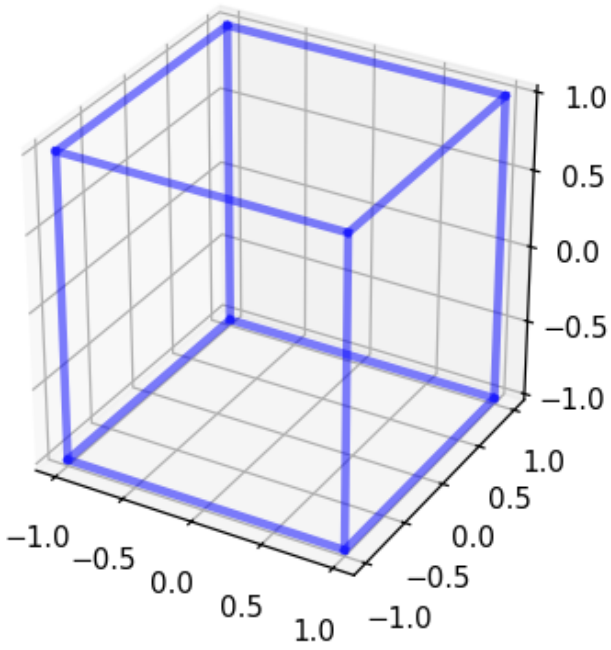


In [39]:
```
from itertools import product, combinations

fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(projection='3d')

#Wuerfel
r = [-1, 1]
for p1, p2 in combinations(np.array(list(product(r, repeat=3))), 2):
    if np.sum(np.abs(p1 - p2)) == r[1] - r[0]:  # falls die Kantenlänge 2 ist
        ax.plot3D(*zip(p1, p2), color="b", alpha=0.5)

ax.set_box_aspect((1, 1, 1))
```



Vektorpfeile in 3d mit **quiver**

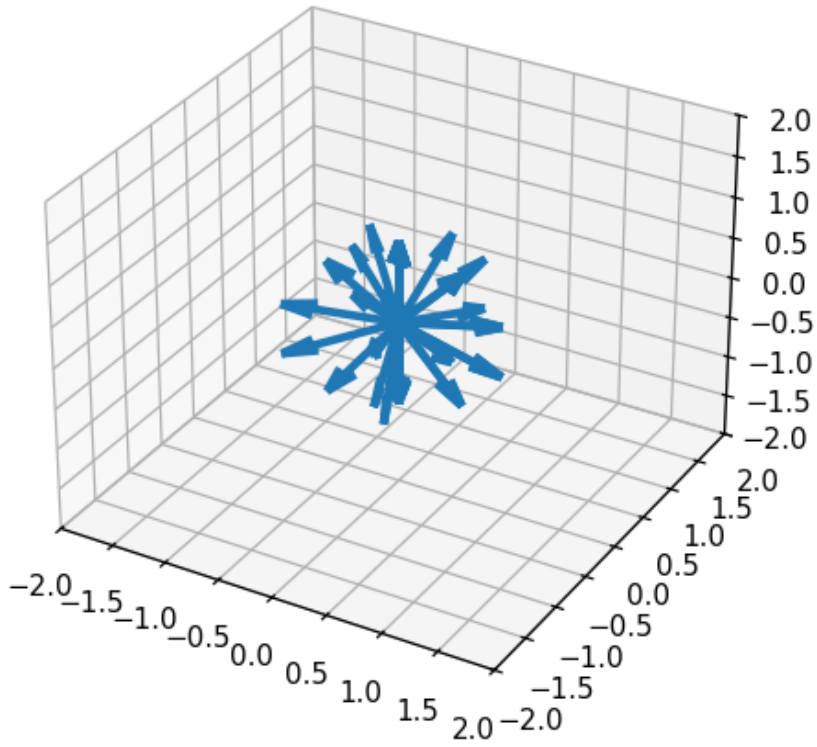
In [41]:

```
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(projection='3d')

phi = np.linspace(0, 2 * np.pi, 6)
phi = np.append(phi, np.nan).reshape(
    -1, 1)  # NaN Trick damit keine ungewollten Linien entstehen
theta = np.linspace(0, np.pi, 6)
theta = np.append(theta, np.nan).reshape(1, -1)

u = np.cos(phi) * np.sin(
    theta)  # durch broadcasting sind x, y, z hier 2D arrays
v = np.sin(phi) * np.sin(theta)
w = np.ones_like(phi) * np.cos(theta)

ax.quiver(0, 0, 0, u, v, w);
```



In [42]:
```
ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)
ax.set_zlim(-2, 2)
```

Out[42]: (-2.0, 2.0)

In [ ]:
```
plt.close('all')
```

In [ ]:
```

```