Deep learning and inverse problems **Exercise 1**:

Problem 1.1

As z is the only random variable,

$$\mathbb{E}_z[\|\hat{x}-x^*\|_2^2]$$

$$= \mathbb{E}_z[(\hat{x}-x^*)^T\cdot(\hat{x}-x^*)]$$

$$= \mathbb{E}_z[\hat{x}^T\cdot\hat{x} - \underbrace{x^{*T}\cdot\hat{x} - \hat{x}^T\cdot x^*}_{\text{Scalar } \Rightarrow\ x^{*T}\cdot\hat{x} = \hat{x}^T\cdot x^*} + x^{*T}\cdot x^*]$$

$$\Downarrow\ \hat{x} = U\cdot U^T\cdot y$$

$$= \mathbb{E}_z[(UU^Ty)^T\cdot(U\cdot U^T\cdot y) - 2\cdot(U\cdot U^T\cdot y)^T\cdot x^*] + x^{*T}\cdot x^*$$

$$= \mathbb{E}_z[y^T\cdot U\cdot \underbrace{U^T\cdot U}_{I}\cdot U^T\cdot y - 2y^T\cdot U\cdot U^T\cdot x^*] + x^{*T}\cdot x^*$$

$$= \mathbb{E}_z[y^T\cdot U\cdot U^T\cdot y - y^T\cdot U\cdot U^T\cdot x^*] - \mathbb{E}_z[y^T\cdot U\cdot U^T x^*] + x^{*T}\cdot x^*$$

$$= \mathbb{E}_z[y^T\cdot U\cdot U^T(y-x^*)] - \mathbb{E}_z[(x^*+z)^T\cdot UU^Tx^*] + x^{*T}\cdot x^*$$

$$= \mathbb{E}_z[(x^*+z)^T\cdot UU^Tz] - \mathbb{E}_z[x^{*T}UU^Tx^* + z^TUU^Tx^*] + x^{*T}x^*$$

$$= \underbrace{\mathbb{E}_z[x^{*T}UU^Tz] - \mathbb{E}_z[zUU^T\cdot x^*]}_{0 \text{ because it's scalar } (x^{*T}UU^Tz)^T = x^{*T}UU^Tz} + \mathbb{E}_z[z^T\cdot UU^T\cdot z] - x^{*T}UU^T\cdot x^* + x^{*T}x^*$$

Now, we are going to consider the term $x^{*T}\cdot x^* - x^{*T}\cdot UU^T\cdot x^*$

Since x^* is in the span by U

$\Rightarrow$ the reprojection of x^{*T} will not change $\|x^*\|_2^2 = x^{*T}\cdot x^*$

$\Rightarrow x^{*T}x^* - x^{*T}UU^Tx^* = 0$.

Proof: assume $x^* = u_1\cdot x_1 + \cdots + u_k\cdot x_k + \underbrace{u_{k+1}\cdot 0 + \cdots + u_n\cdot 0}_{\text{since } x^* \text{ is in } k\text{-subspace}}$

$$\Rightarrow U^T\cdot x^* = \begin{bmatrix} u_1^T \\ \vdots \\ u_k^T \end{bmatrix}\cdot x^* = \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}$$

orthonormal basis: $\begin{cases} u_1^T\cdot u_1 = 1. \\ u_1^T\cdot u_i = 0 \quad (i\neq 1) \end{cases}$

$$\Rightarrow (U^T\cdot x^*)^T\cdot(U^Tx^*) = \sum_{i=1}^{k} x_i^2 = x^{*T}\cdot x^*$$

$$\Rightarrow (U^T\cdot x^*)^T\cdot(U^Tx^*) = x^{*T}\cdot x^*$$

Now let's see $\mathbb{E}_z[z^T\cdot UU^Tz]$.

Proof: $\mathbb{E}_z[z^T\cdot U\cdot U^Tz] = \mathbb{E}_z[\|\hat{x}-x^*\|_2^2]$

Similarly, assume $z = u_1z_1 + \cdots + u_kz_k + u_{k+1}z_{k+1} + \cdots + u_nz_n$

$$U^T\cdot z = \begin{bmatrix} u_1^T \\ \vdots \\ u_k^T \end{bmatrix}\cdot z = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix}$$

$$\Rightarrow \mathbb{E}_z[z^T\cdot UU^Tz] = \mathbb{E}_z[(U^Tz)^T\cdot U^Tz] = \mathbb{E}_z[\sum_{i=1}^{k} z_i^2]$$

from the covariance of z $\Rightarrow \mathbb{E}_z[z_i^2] = \frac{\sigma^2}{n}$.

$$\Rightarrow \mathbb{E}_z[\sum_{i=1}^{k} z_i^2] = \frac{k\cdot\sigma^2}{n} \quad ▨$$

Problem 1.2

- If the dimension of subspace reduces, this algorithm will denoise more (better performance). Except for the extreme case like dimension = 1.

- A better algorithm exists ?

  For low-dimensional signal models, it is possible to solve inverse problems.

  But linear subspaces are not a good model.

  More suitable solutions may be: sparse models,
  models in form of neural networks ...

  [reference: lecture notes]

Problem 1.3

In [1]:
```
import numpy as np
from scipy.linalg import orth
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
def main(k, n, num_point, var_noise):
    # Step 1: Generate a random k-dim subspace based on the orthogonal basis U
    original_space = np.random.random(size=(n, n))
    U_full = orth(original_space)
    indices = np.random.choice(np.arange(1000), size=k, replace=False)
    U = U_full[:, indices]

    # Step 2: Generate original x
    c = np.random.rand(k, num_point)
    x_original = np.matmul(U, c)

    # Step 3: Add Gaussian zero-mean noise to obtain noisy observations
    mean = np.zeros(n)
    cov = (var_noise / n) * np.identity(n)
    noise_all = np.zeros((n, num_point))
    for point in range(num_point):
        noise = np.random.multivariate_normal(mean, cov, 1)
        noise_all[:, point] = np.reshape(noise, n)
    y = x_original + noise_all

    # Step 4: Denoise and get the estimation
    x_estimate = np.matmul(np.matmul(U, U.T), y)

    # Step 5: Calculate the MSE
    diff = x_estimate - x_original
    square = np.square(diff)
    ssd = np.sum(np.square(diff), axis=0)
    norm = np.sum(np.square(x_original), axis=0)
    mse_avg = ssd / norm
    return mse_avg
```

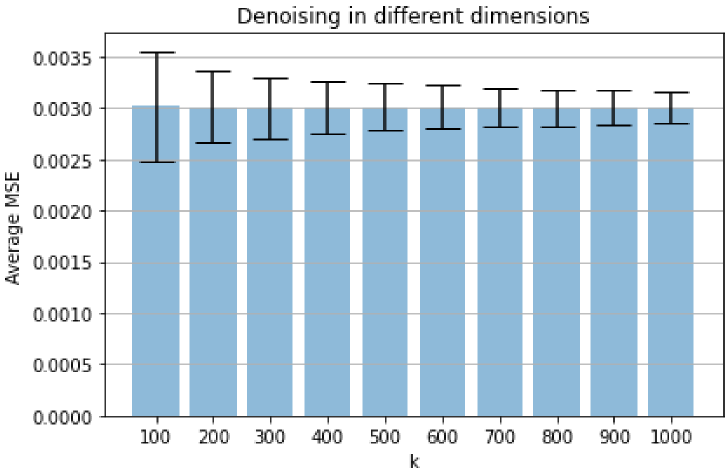
In [3]:
```
# Experiment 1: Dimension k ranging from 100 to 1000
k_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000] # dim of the subspace
n = 1000 # dim of the original space
num_point = 500
variance_noise = 1

list_mse_mean = []
list_mse_std = []

for k in k_list:
    mse_avg = main(k, n, num_point, variance_noise)
    list_mse_mean.append(np.mean(mse_avg))
    list_mse_std.append(np.std(mse_avg))

# Plot
label_k = ['100', '200', '300', '400', '500', '600', '700', '800', '900', '1000']
x = np.arange(len(label_k))
fig, ax = plt.subplots()
ax.bar(x, list_mse_mean,
       yerr=list_mse_std,
       align='center',
       alpha=0.5,
       ecolor='black',
       capsize=10)
ax.set_ylabel('Average MSE')
ax.set_xticks(x)
ax.set_xticklabels(label_k)
ax.set_xlabel('k')
ax.set_title('Denoising in different dimensions')
ax.yaxis.grid(True)
```

```
plt.tight_layout()
plt.show()
```



In [4]:

```
# Experiment 2: Dimension k ranging from 1 to 1000
k_list = [1, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000] # dim of the subspace
n = 1000 # dim of the original space
num_point = 500
variance_noise = 1

list_mse_mean = []
list_mse_std = []

for k in k_list:
    mse_avg = main(k, n, num_point, variance_noise)
    list_mse_mean.append(np.mean(mse_avg))
    list_mse_std.append(np.std(mse_avg))

# Plot
label_k = ['1', '100', '200', '300', '400', '500', '600', '700', '800', '900', '1000']
x = np.arange(len(label_k))
fig, ax = plt.subplots()
ax.bar(x, list_mse_mean,
       yerr=list_mse_std,
       align='center',
       alpha=0.5,
       ecolor='black',
       capsize=10)
ax.set_ylabel('Average MSE')
ax.set_xticks(x)
ax.set_xticklabels(label_k)
ax.set_xlabel('k')
ax.set_title('Denoising in different dimensions')
ax.yaxis.grid(True)

plt.tight_layout()
plt.show()
```