

hw2__code__solution

October 16, 2024

1 Math 76 HW2, Fall 2024

Name:

For all plots, make sure to include a title, x-axis label, and y-axis label.

```
[2]: import numpy as np
import matplotlib.pyplot as plt
from scipy.special import gamma as gamma_func

from hw2_helper_funcs import *
```

2 Problem 3.5

2.1 Part A

Derive the formula

$$a_{ij} = \frac{d}{n} (d^2 + ((i-j)/n)^2)^{-3/2}, \quad i, j = 1, \dots, n \quad (1)$$

for the matrix elements.

Response: *(it may be easier to do this on paper and submit it alongside the notebook)*

2.2 Part B

Compute $\mathbf{b} = \mathbf{A} \mathbf{x}$ for two different exact solutions x : (1) a smooth solution (not a constant or linear vector) and (2) a vector with one or more jump discontinuities. Plot both vectors and label them.

```
[3]: # Parameters
n = 32
d = 0.3

# Smooth x_true
grid = np.linspace(0,100,n)
x_smooth = np.sin(grid/4) + 0.1*grid

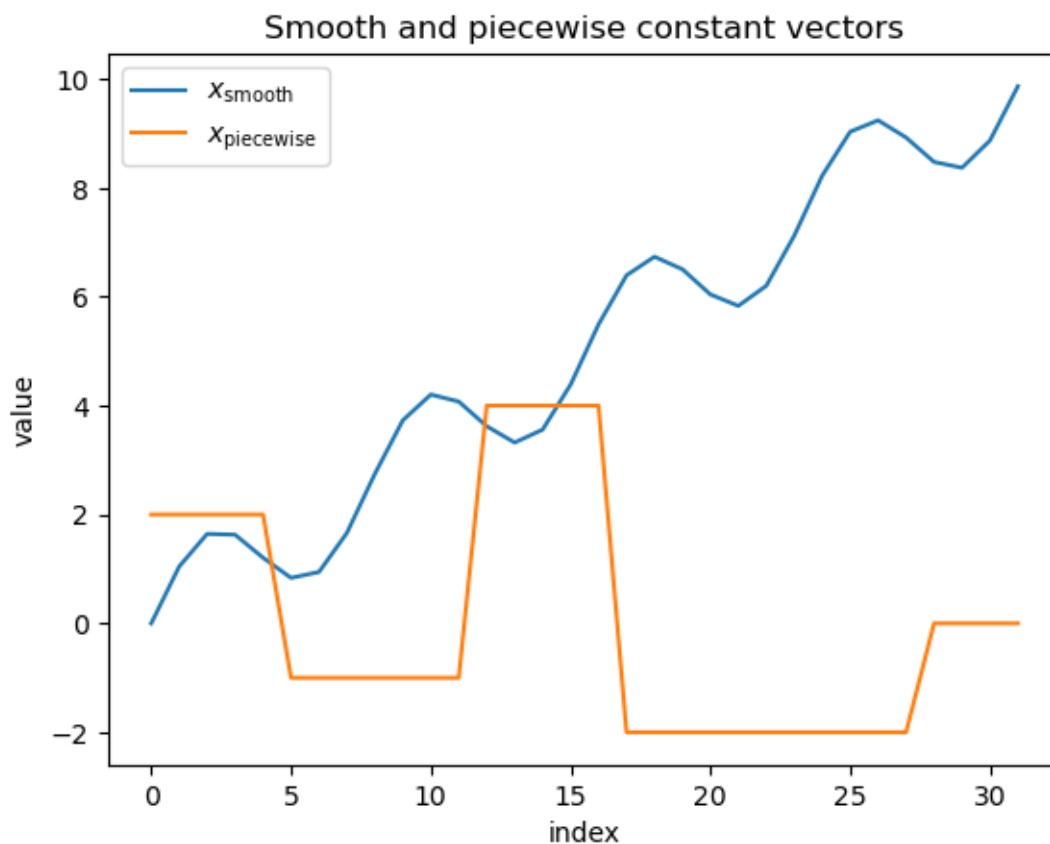
# Piecewise constant x_true
x_piecewise_constant = np.zeros(n)
```

```
x_piecewise_constant[:5] = 2.0
x_piecewise_constant[5:12] = -1.0
x_piecewise_constant[12:17] = 4.0
x_piecewise_constant[17:28] = -2.0
```

```
# Build matrix
```

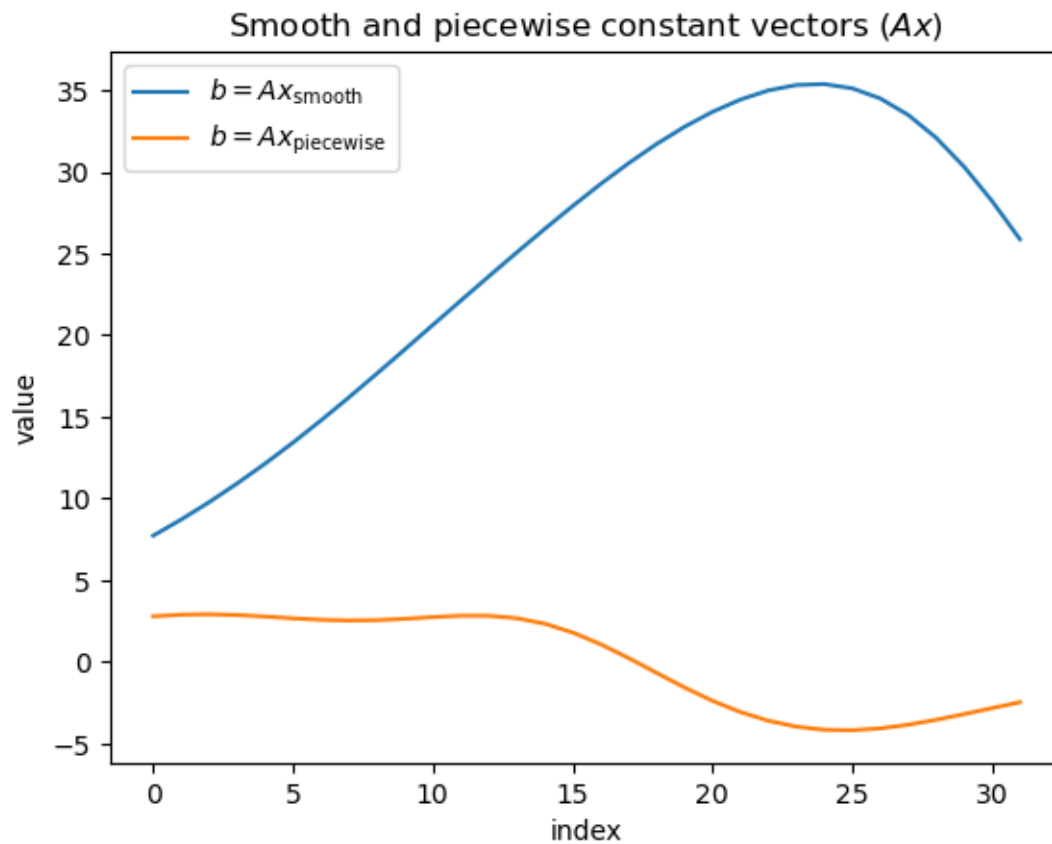
```
A = build_gravity_matrix(n,d)
```

```
[4]: plt.plot(x_smooth, label="$x_{\\text{smooth}}$")
plt.plot(x_piecewise_constant, label="$x_{\\text{piecewise}}$")
plt.xlabel("index")
plt.ylabel("value")
plt.title("Smooth and piecewise constant vectors")
plt.legend()
plt.show()
```



```
[5]: # Get exact b's
b_smooth = A @ x_smooth
b_piecewise_constant = A @ x_piecewise_constant
```

```
[6]: plt.plot(b_smooth, label="$b = A x_{\\text{smooth}}$")
plt.plot(b_piecewise_constant, label="$b = A x_{\\text{piecewise}}$")
plt.xlabel("index")
plt.ylabel("value")
plt.title("Smooth and piecewise constant vectors ($A x$)")
plt.legend()
plt.show()
```



3 Part C

Study how the RHS vectors change with the depth parameter d . *Suggestion: plot the rhs vectors b for varying parameter d on the same plot. Make sure to label each curve with the value of d that was used.*

3.0.1 For smooth x_{true}

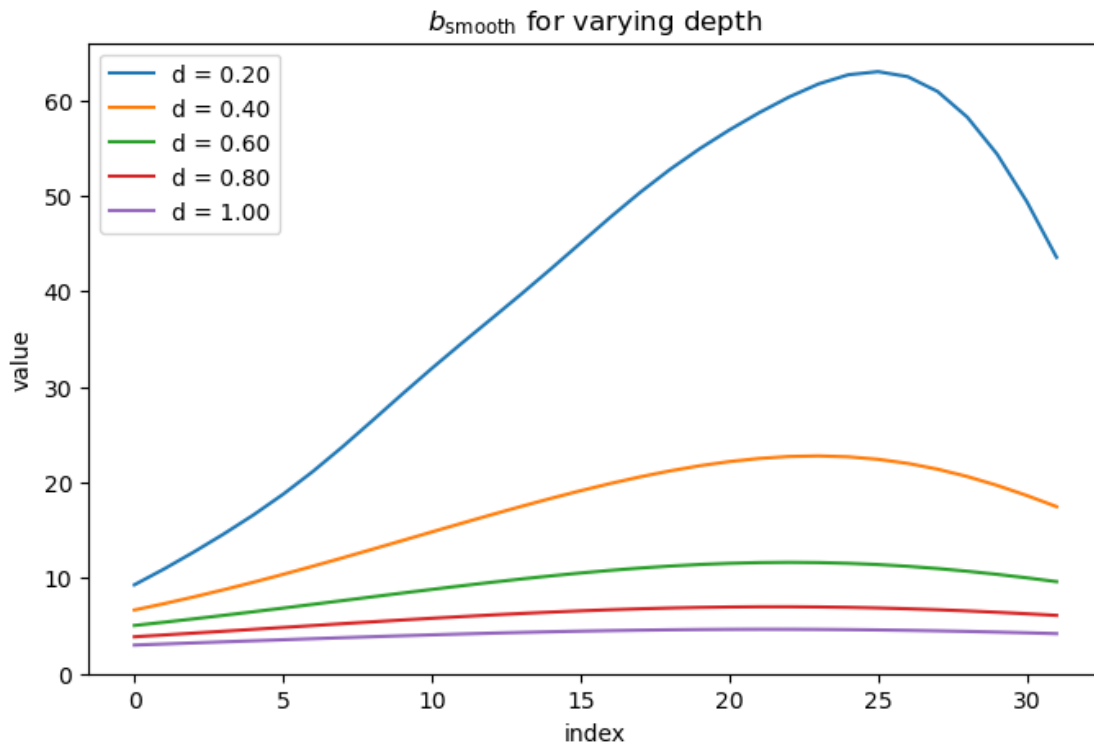
```
[7]: ds = np.arange(0.2, 1.1, 0.2)
plt.figure(figsize=(8,5))
for d in ds:

    # Build A matrix
    A = build_gravity_matrix(n,d)

    # Get rhs b
    b_smooth = A @ x_smooth

    plt.plot(b_smooth, label=f"d = {d:.2f}")

plt.xlabel("index")
plt.ylabel("value")
plt.title("$b_{\text{smooth}}$ for varying depth")
plt.legend()
plt.show()
```



3.0.2 For piecewise constant x_{true}

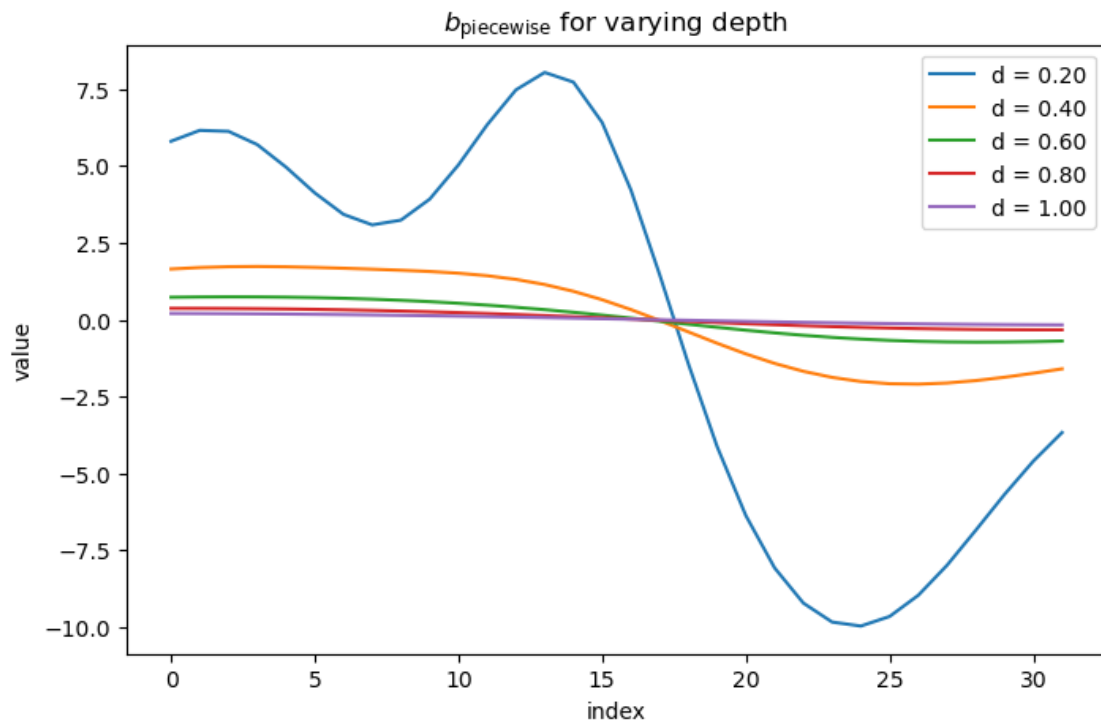
```
[8]: ds = np.arange(0.2, 1.1, 0.2)
plt.figure(figsize=(8,5))
for d in ds:

    # Build A matrix
    A = build_gravity_matrix(n,d)

    # Get rhs b
    b_piecewise_constant = A @ x_piecewise_constant

    plt.plot(b_piecewise_constant, label=f"d = {d:.2f}")

plt.xlabel("index")
plt.ylabel("value")
plt.title("$b_{\\text{piecewise}}$ for varying depth")
plt.legend()
plt.show()
```



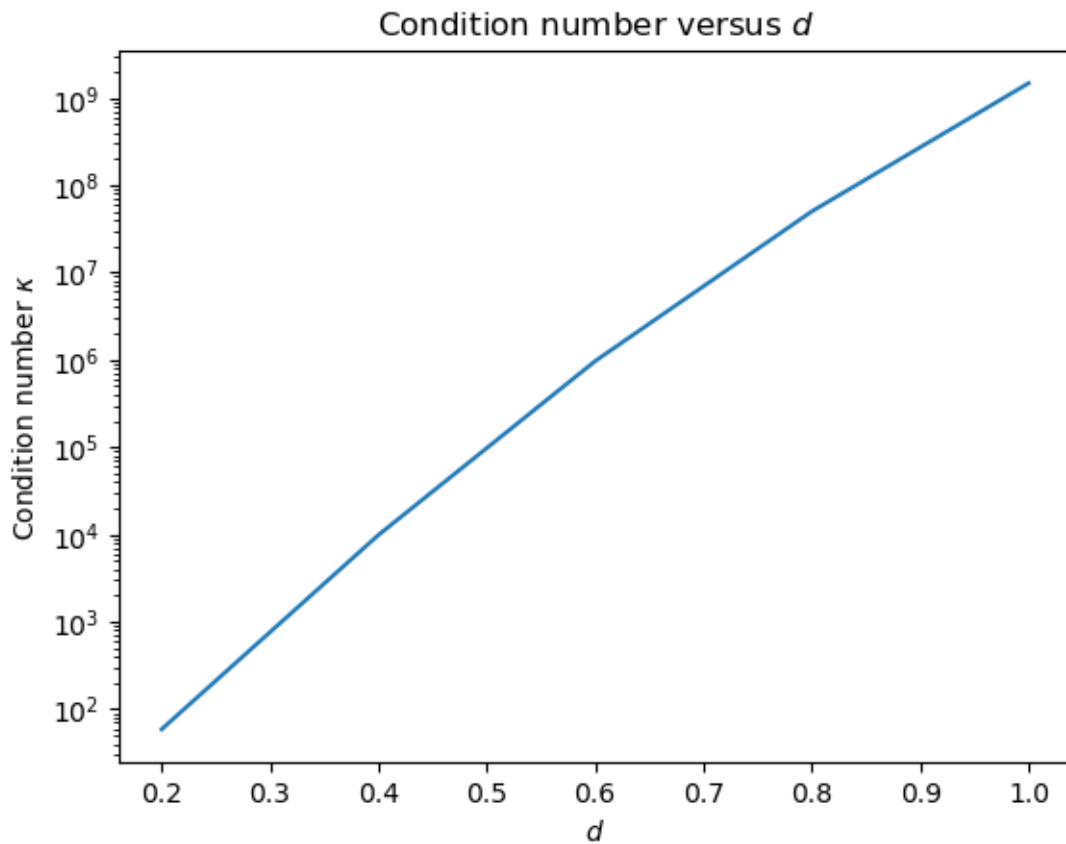
3.1 Part D

Study how the condition number of A varies for varying d and fixed n ; then, study how the condition number of A varies for varying n but fixed d .

3.1.1 Varying d , fixed n

```
[9]: n = 10
ds = np.arange(0.2, 1.1, 0.2)
conds = []
for d in ds:
    A = build_gravity_matrix(n,d)
    cond_A = np.linalg.cond(A)
    conds.append(cond_A)
```

```
[10]: plt.semilogy(ds, conds)
plt.xlabel("$d$")
plt.ylabel("Condition number  $\kappa$ ")
plt.title("Condition number versus $d$")
plt.show()
```

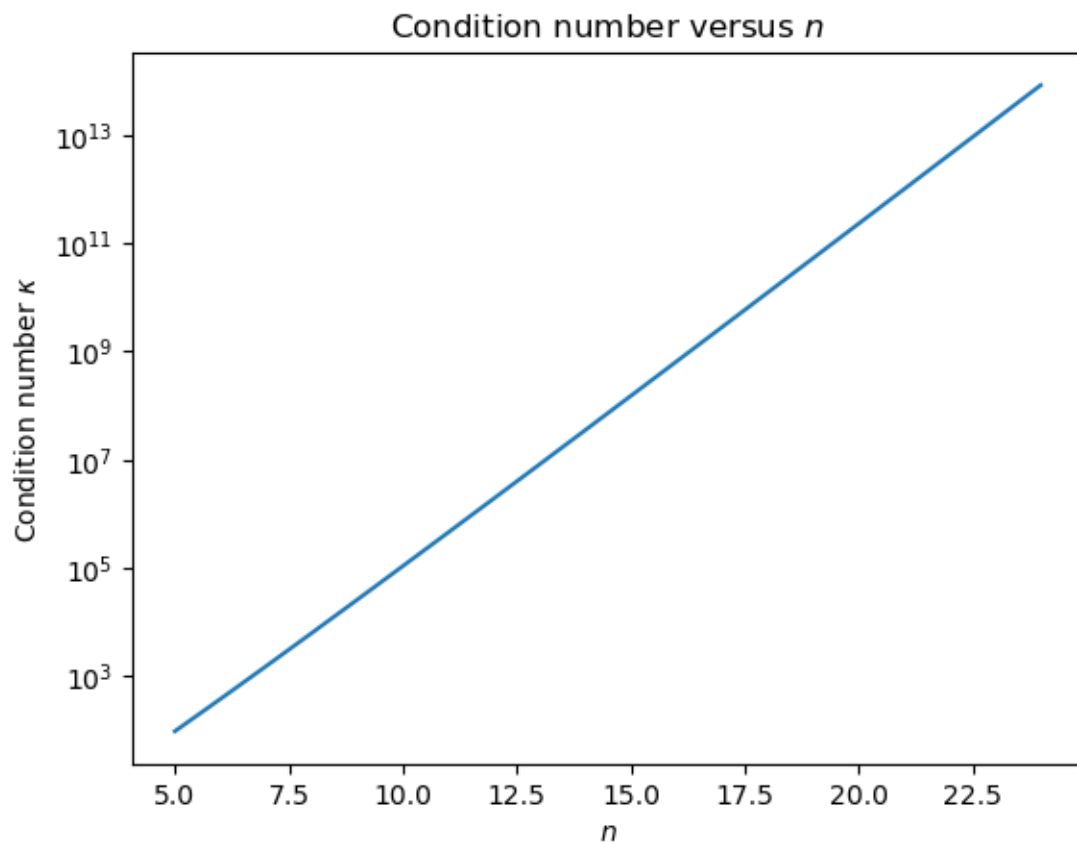


Explanation of observed behavior:

3.1.2 Varying n , fixed d

```
[11]: d = 0.5
ns = np.arange(5, 25)
conds = []
for n in ns:
    A = build_gravity_matrix(n,d)
    cond_A = np.linalg.cond(A)
    conds.append(cond_A)
```

```
[12]: plt.semilogy(ns, conds)
plt.xlabel("$n$")
plt.ylabel("Condition number $\kappa$")
plt.title("Condition number versus $n$")
plt.show()
```



Explanation of observed behavior:

3.2 Part E

Solve the problem $Ax = b$ using a noise-free rhs b . Then, try again but instead solve $Ax = b + \delta$ where δ is a white noise Gaussian vector $\delta \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. Set $n = 32$.

```
[13]: # Parameters
n = 32
d = 0.3

# Smooth x_true
grid = np.linspace(0,100,n)
x_smooth = np.sin(grid/4) + 0.1*grid

# Piecewise constant x_true
x_piecewise_constant = np.zeros(n)
x_piecewise_constant[:5] = 2.0
x_piecewise_constant[5:12] = -1.0
x_piecewise_constant[12:17] = 4.0
x_piecewise_constant[17:28] = -2.0

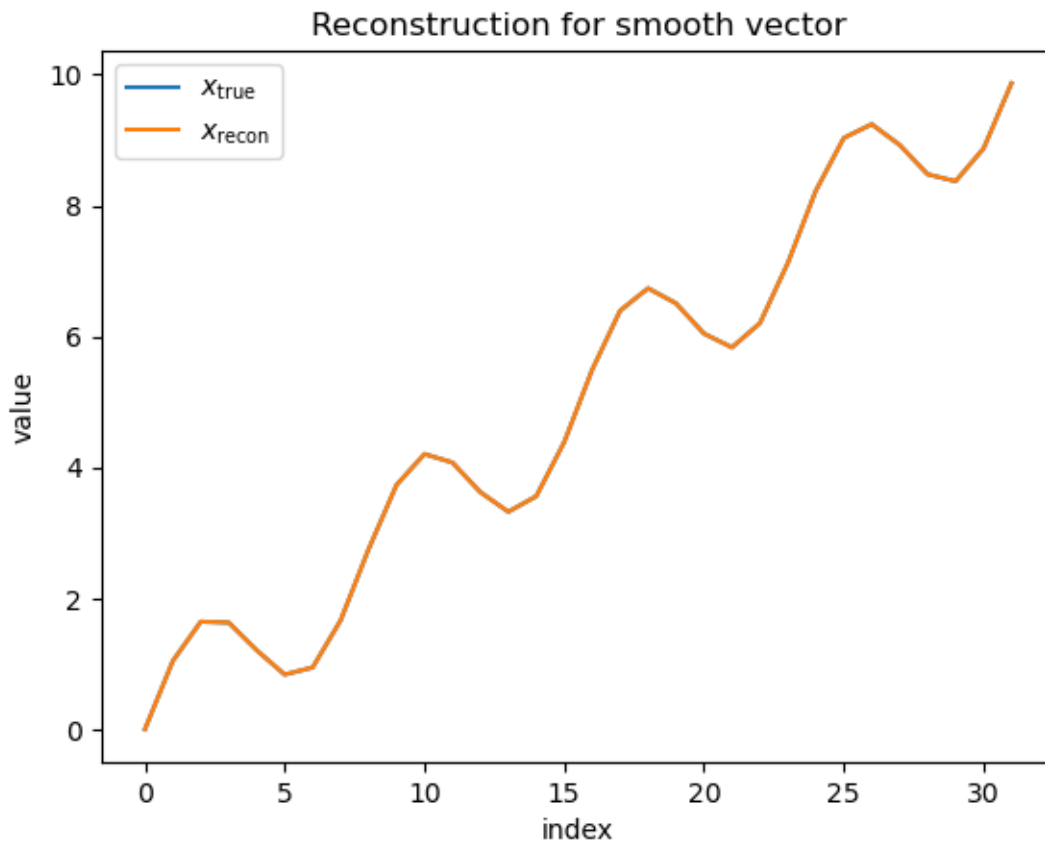
# Build matrix
A = build_gravity_matrix(n,d)

# Get exact b's
b_smooth = A @ x_smooth
b_piecewise_constant = A @ x_piecewise_constant
```

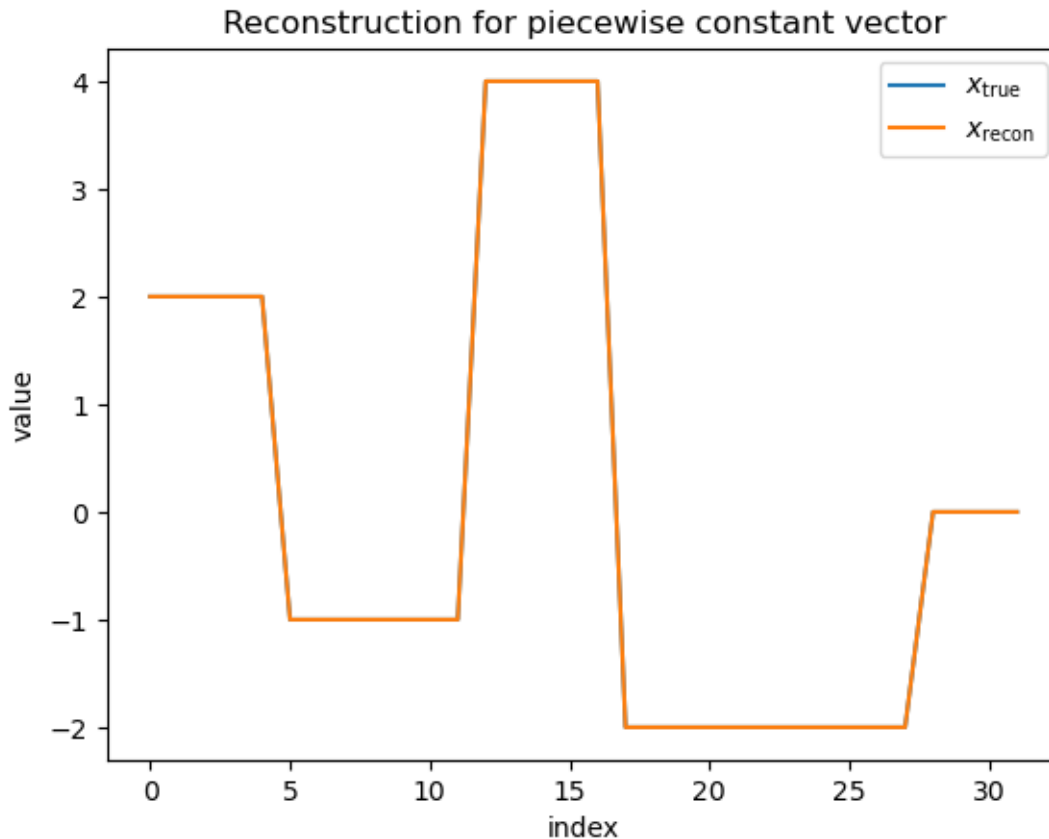
Solve $Ax = b$ (for both smooth and piecewise constant vectors)

```
[14]: # Solve
x_smooth_est = np.linalg.solve(A, b_smooth)
x_piecewise_constant_est = np.linalg.solve(A, b_piecewise_constant)
```

```
[15]: plt.plot(x_smooth, label="$x_{\\text{true}}$")
plt.plot(x_smooth_est, label="$x_{\\text{recon}}$")
plt.legend()
plt.title("Reconstruction for smooth vector")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```

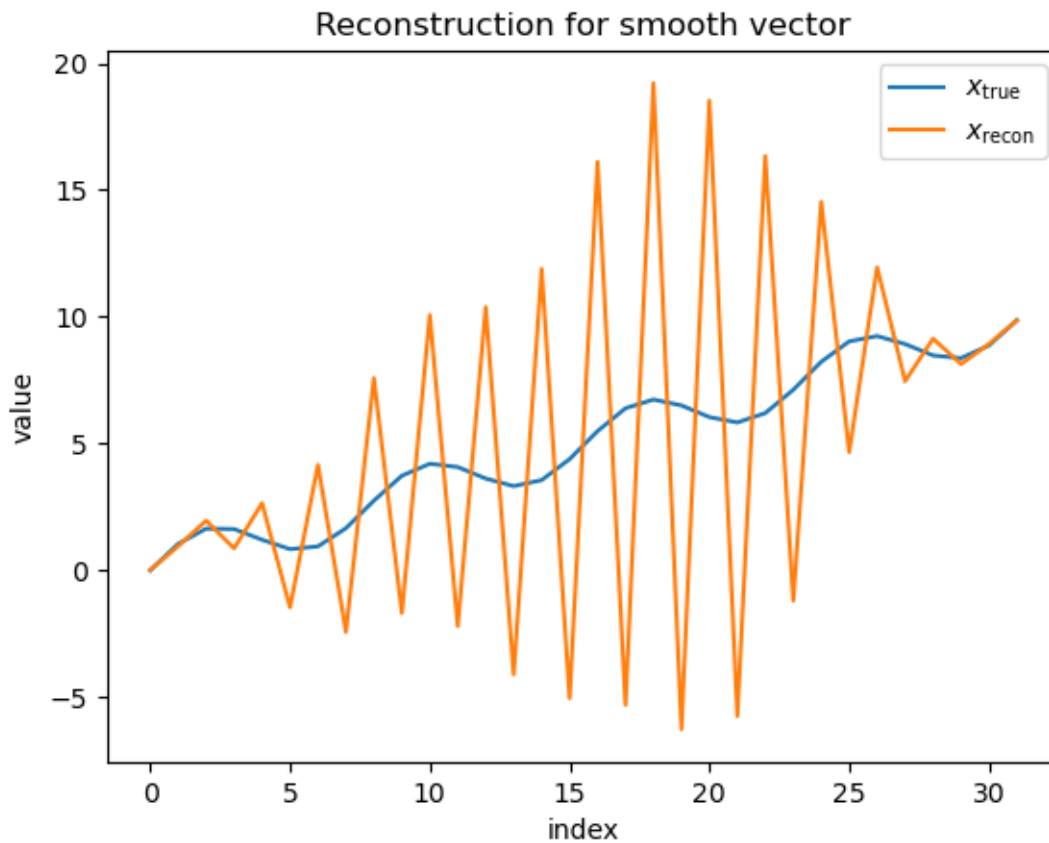
```
[16]: plt.plot(x_piecewise_constant, label="$x_{\\text{true}}$")
plt.plot(x_piecewise_constant_est, label="$x_{\\text{recon}}$")
plt.legend()
plt.title("Reconstruction for piecewise constant vector")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```



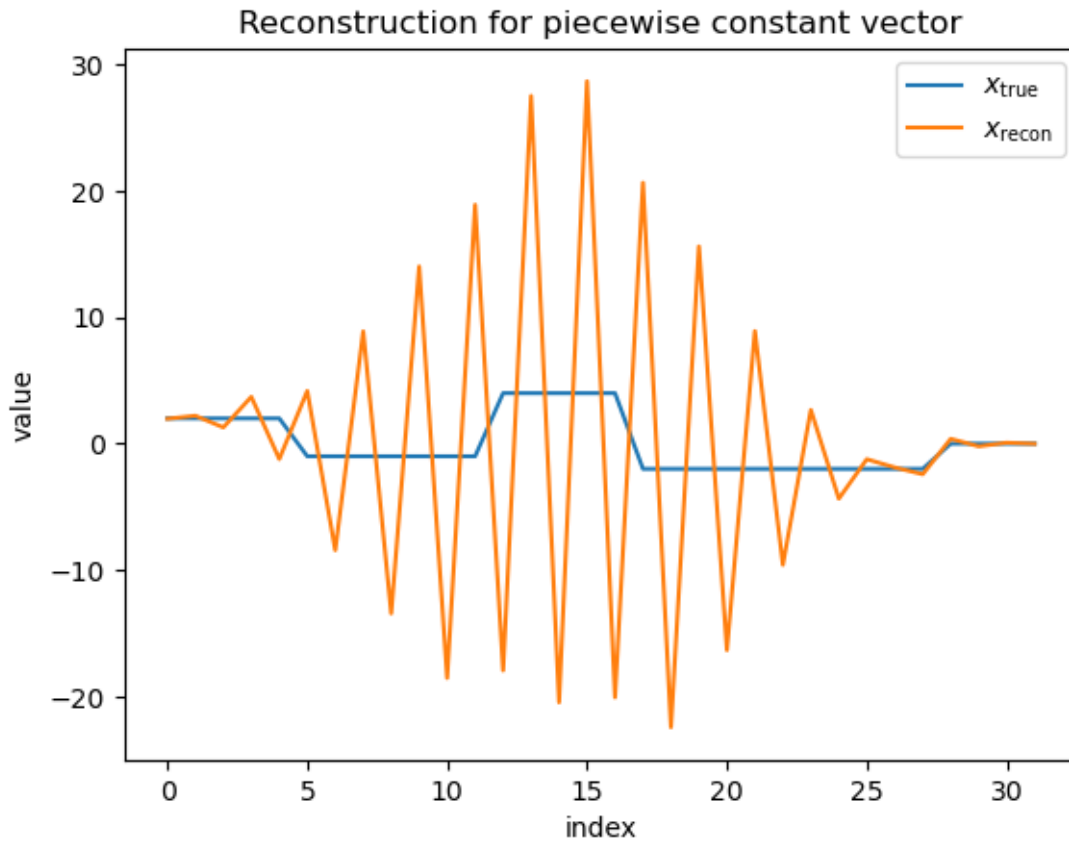
Solve $Ax = b + \delta$ (for both smooth and piecewise constant vectors)

```
[17]: # Solve
x_smooth_est = np.linalg.solve(A, b_smooth + 1e-9*np.random.
    ↪normal(size=len(b_smooth)))
x_piecewise_constant_est = np.linalg.solve(A, b_piecewise_constant + 1e-9*np.
    ↪random.normal(size=len(b_piecewise_constant)))
```

```
[18]: plt.plot(x_smooth, label="$x_{\\text{true}}$")
plt.plot(x_smooth_est, label="$x_{\\text{recon}}$")
plt.legend()
plt.title("Reconstruction for smooth vector")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```



```
[19]: plt.plot(x_piecewise_constant, label="$x_{\\text{true}}$")
plt.plot(x_piecewise_constant_est, label="$x_{\\text{recon}}$")
plt.legend()
plt.title("Reconstruction for piecewise constant vector")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```



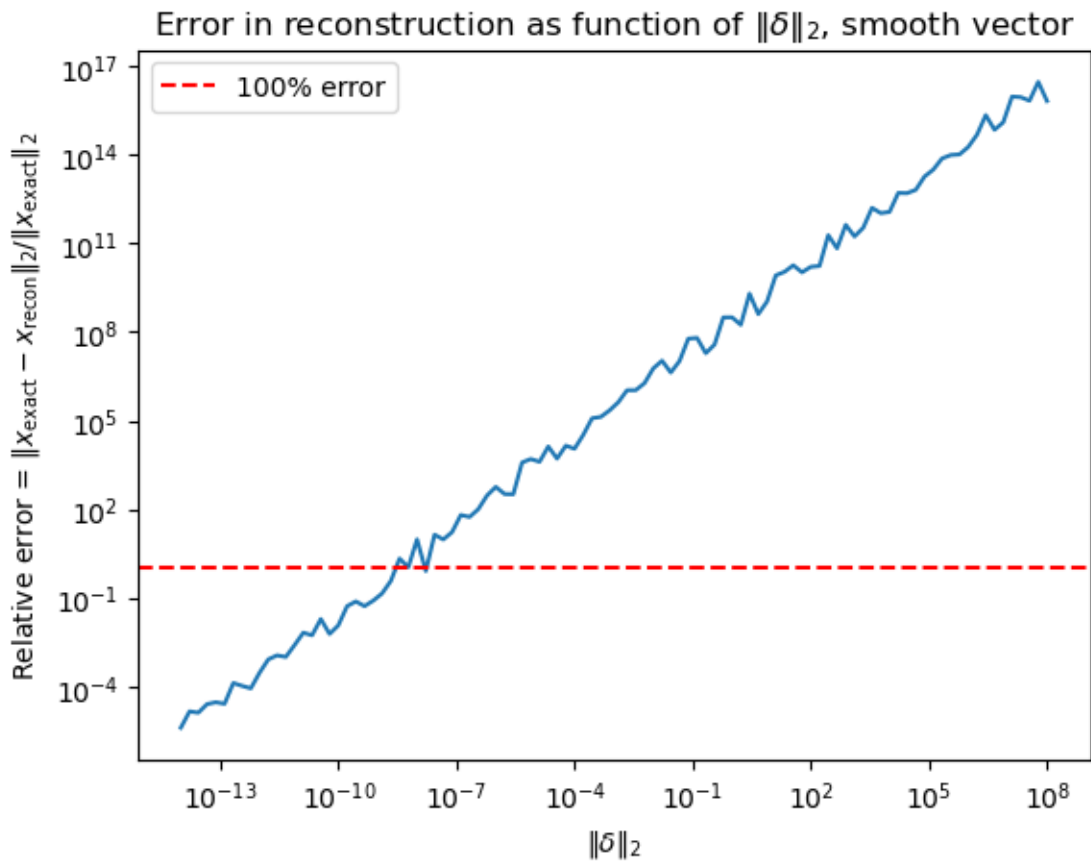
How large can $\|\delta\|_2$ get before the inverted noise starts to dominate? Design a computational study/visualization below to support your claim.

Response:

```
[20]: np.random.seed(0)
delta_norms = np.logspace(-14, 8, 100)
x_errs = []
x_rel_errs = []
x_exact_norm = np.linalg.norm(x_smooth)
for delta_norm in delta_norms:
    randvec = np.random.normal(size=n)
    b_rand = b_smooth + (randvec/np.linalg.norm(randvec))*delta_norm

    x_est = np.linalg.solve(A, b_rand)
    x_err = np.linalg.norm(x_est - x_smooth)
    x_errs.append(x_err)
    x_rel_errs.append(x_err/x_exact_norm)
```

```
[21]: plt.loglog(delta_norms, x_rel_errs)
plt.xlabel("$\|\delta\|_2$")
plt.ylabel("Relative error = $\|x_{\text{exact}} - x_{\text{recon}}\|_2 / \|x_{\text{exact}}\|_2$")
plt.title("Error in reconstruction as function of $\|\delta\|_2$, smooth vector")
plt.axhline(1, color="red", ls="--", label="100% error")
plt.legend()
plt.show()
```



```
[22]: np.random.seed(0)
delta_norms = np.logspace(-14, 8, 100)
x_errs = []
x_rel_errs = []
x_exact_norm = np.linalg.norm(x_piecewise_constant)
for delta_norm in delta_norms:
    randvec = np.random.normal(size=n)
    b_rand = b_piecewise_constant + (randvec/np.linalg.norm(randvec))*delta_norm
    x_est = np.linalg.solve(A, b_rand)
```

```

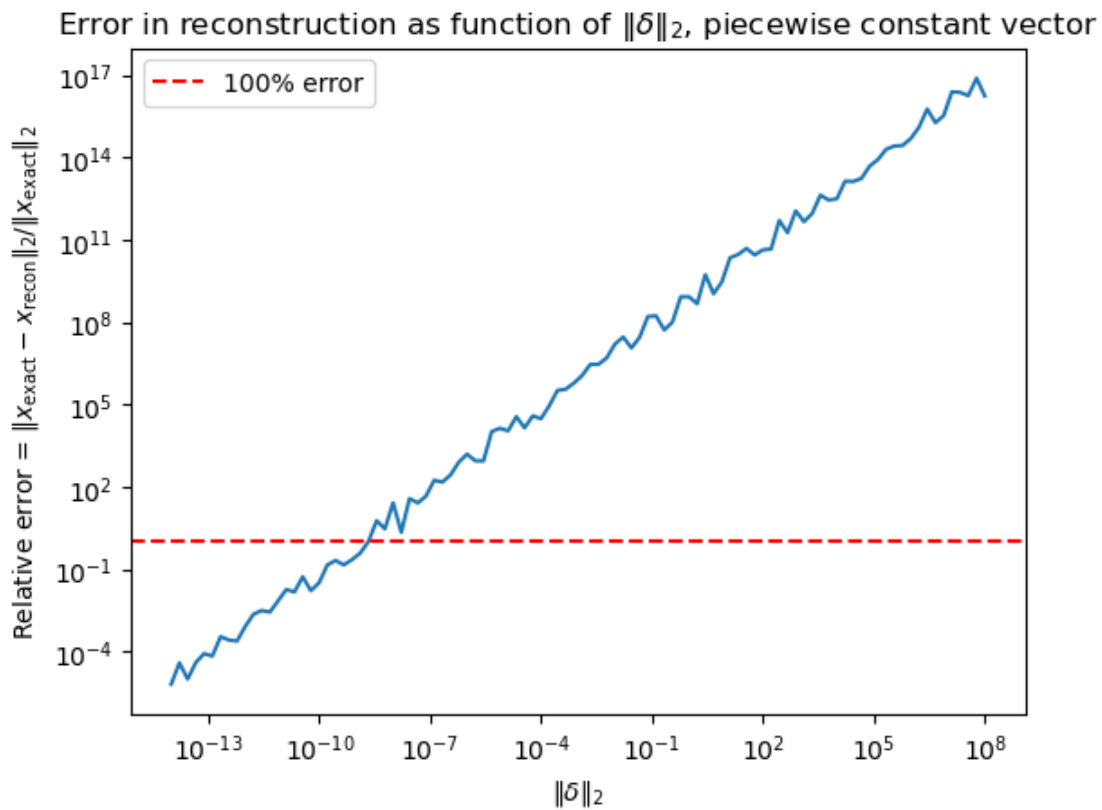
x_err = np.linalg.norm(x_est - x_piecewise_constant)
x_errs.append(x_err)
x_rel_errs.append(x_err/x_exact_norm)

```

```

[23]: plt.loglog(delta_norms, x_rel_errs)
plt.xlabel("$\|\delta\|_2$")
plt.ylabel("Relative error = $\|x_{\text{exact}} - x_{\text{recon}}\|_2 / \|x_{\text{exact}}\|_2$")
plt.title("Error in reconstruction as function of $\|\delta\|_2$, piecewise_
↪ constant vector")
plt.axhline(1, color="red", ls="--", label="100% error")
plt.legend()
plt.show()

```



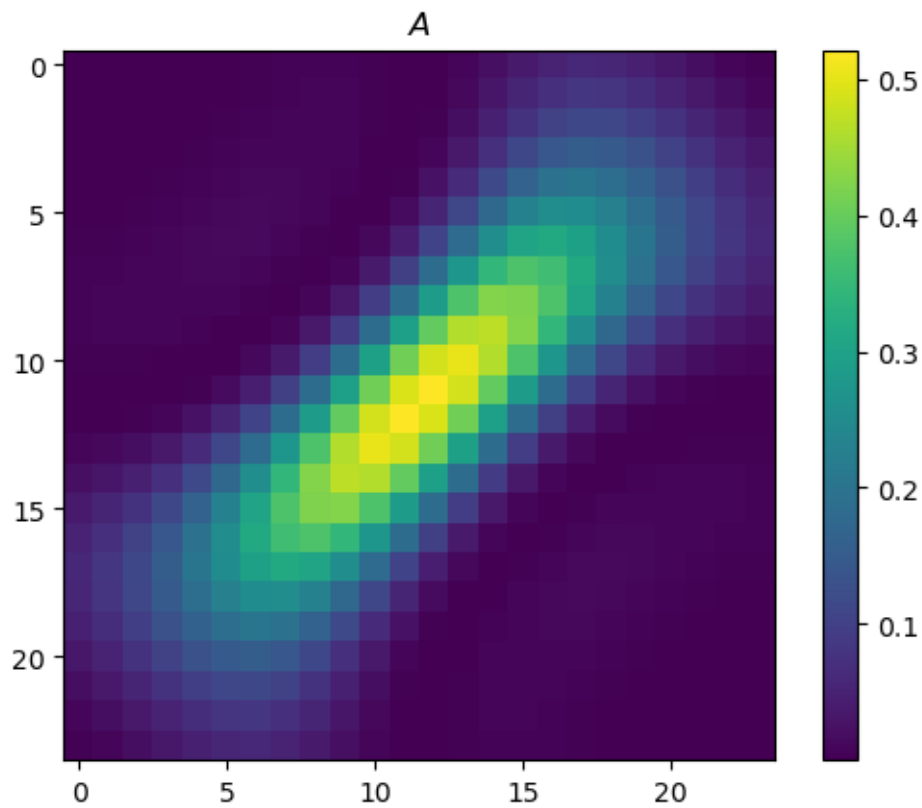
4 Problem 3.6

4.1 Part A

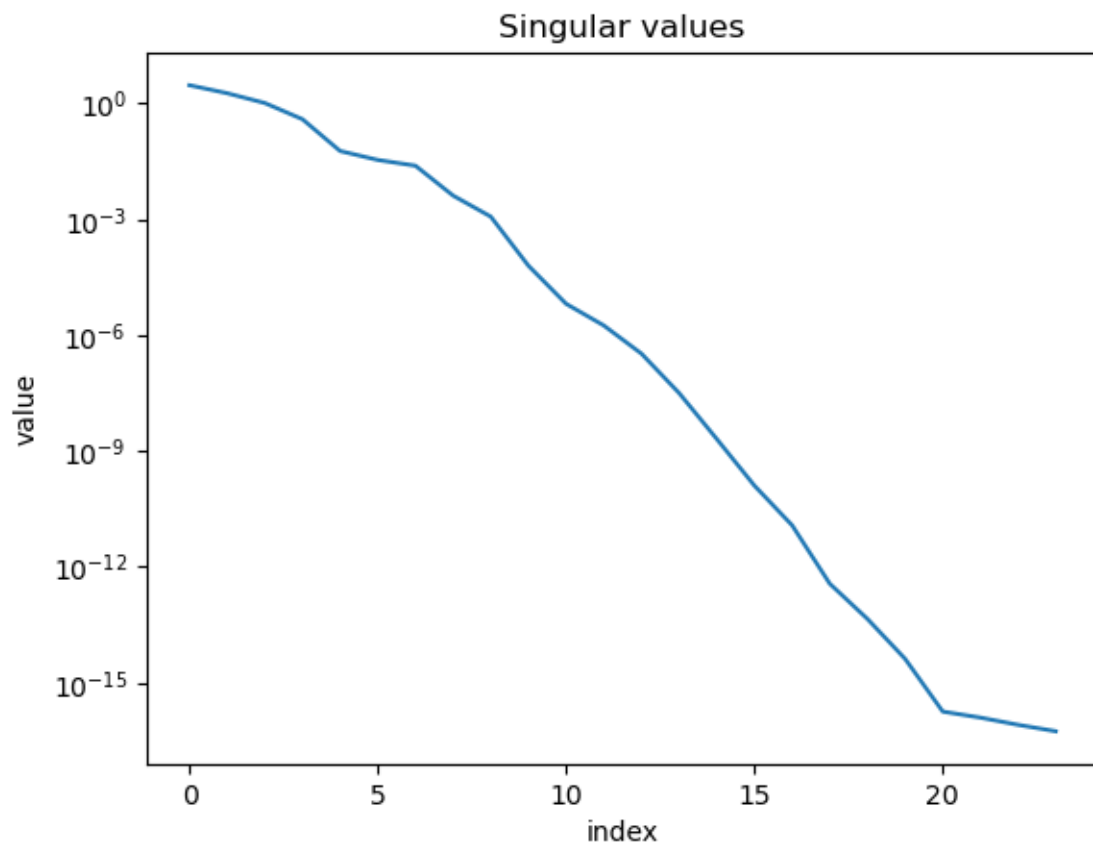
Generate the problem.

```
[24]: n = 24
A, b_exact, x_exact = shaw(n)
svdres = np.linalg.svd(A)
U, s, Vt = svdres.U, svdres.S, svdres.Vh
S = np.diag(s)
V = Vt.T
```

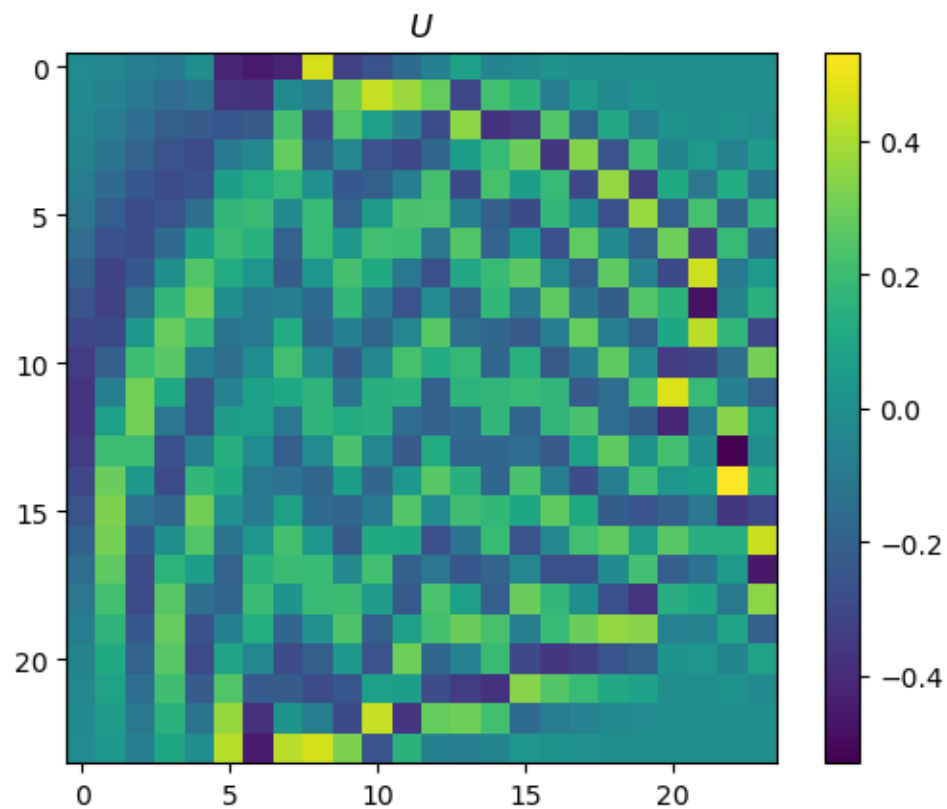
```
[25]: plt.imshow(A)
plt.title("$A$")
plt.colorbar()
plt.show()
```



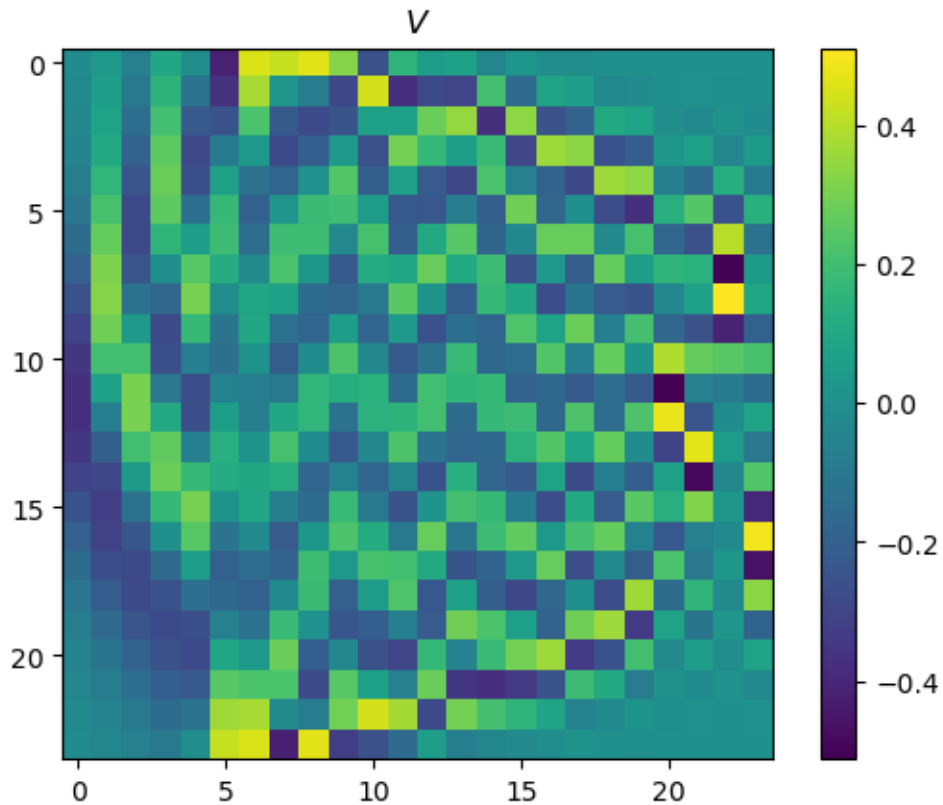
```
[26]: plt.semilogy(s)
plt.title("Singular values")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```



```
[27]: plt.imshow(U)
plt.title("$U$")
plt.colorbar()
plt.show()
```

```
[28]: plt.imshow(V)
plt.title("$V$")
plt.colorbar()
plt.show()
```

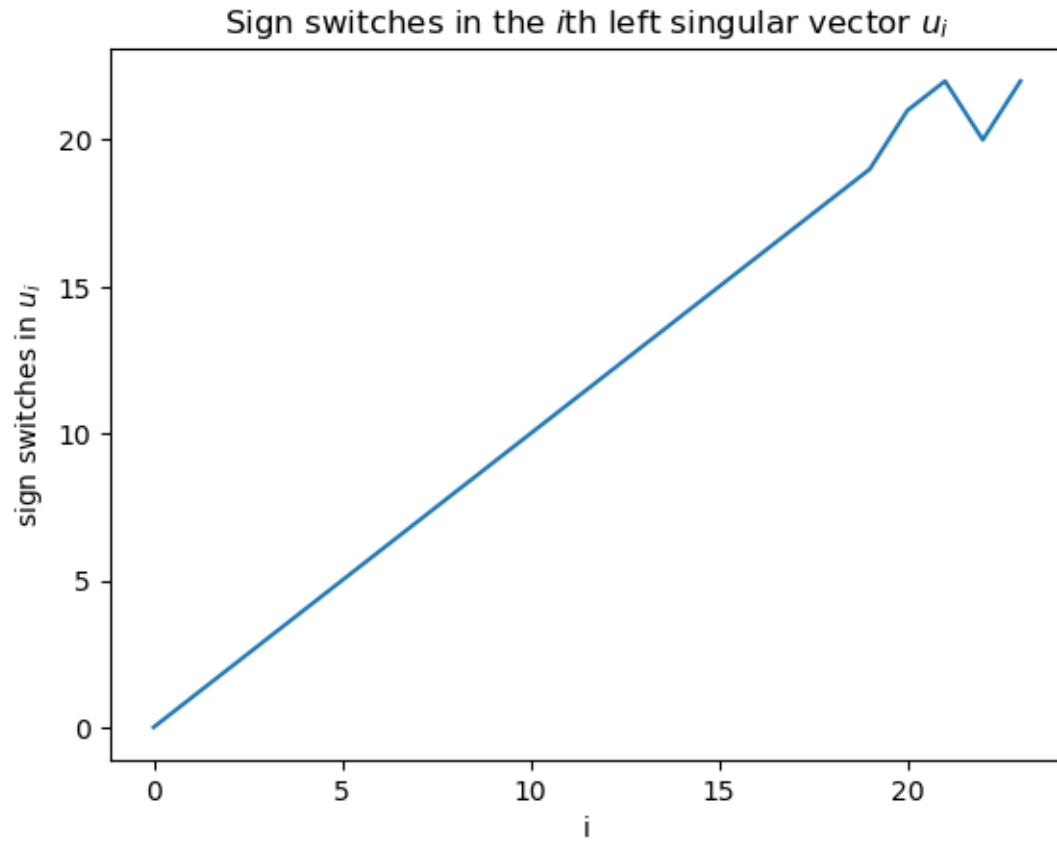


What can be said about the number of sign changes in the left and right singular vectors? Try computing the number of sign changes in each vector and plotting them as a function of i . For convenience, you can use the function `count_sign_switches` in `hw2_helper_funcs.py`.

Response:

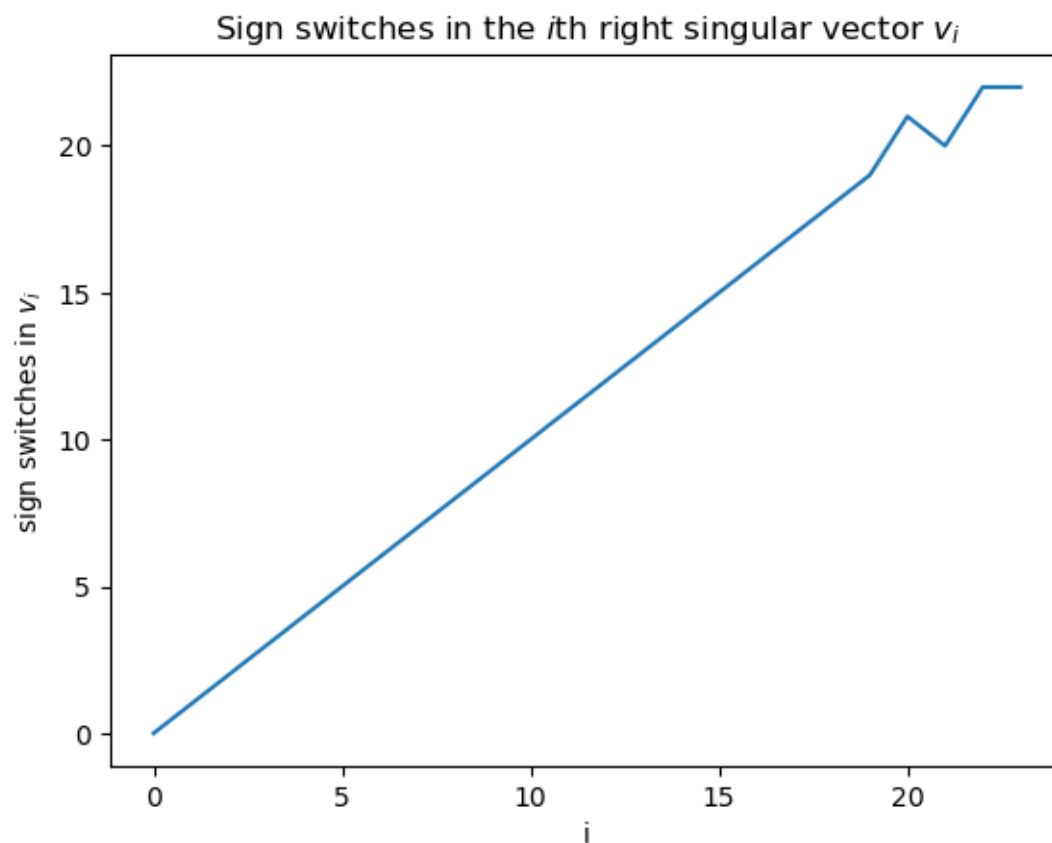
```
[29]: switches = []
      for i in range(n):
          switches.append( count_sign_switches(U[:,i]) )

      plt.title("Sign switches in the  $i$ th left singular vector  $u_i$ ")
      plt.plot(switches)
      plt.xlabel("i")
      plt.ylabel("sign switches in  $u_i$ ")
      plt.show()
```



```
[30]: switches = []
      for i in range(n):
          switches.append( count_sign_switches(V[:,i].T) )

      plt.title("Sign switches in the $i$th right singular vector $v_i$")
      plt.plot(switches)
      plt.xlabel("i")
      plt.ylabel("sign switches in $v_i$")
      plt.show()
```

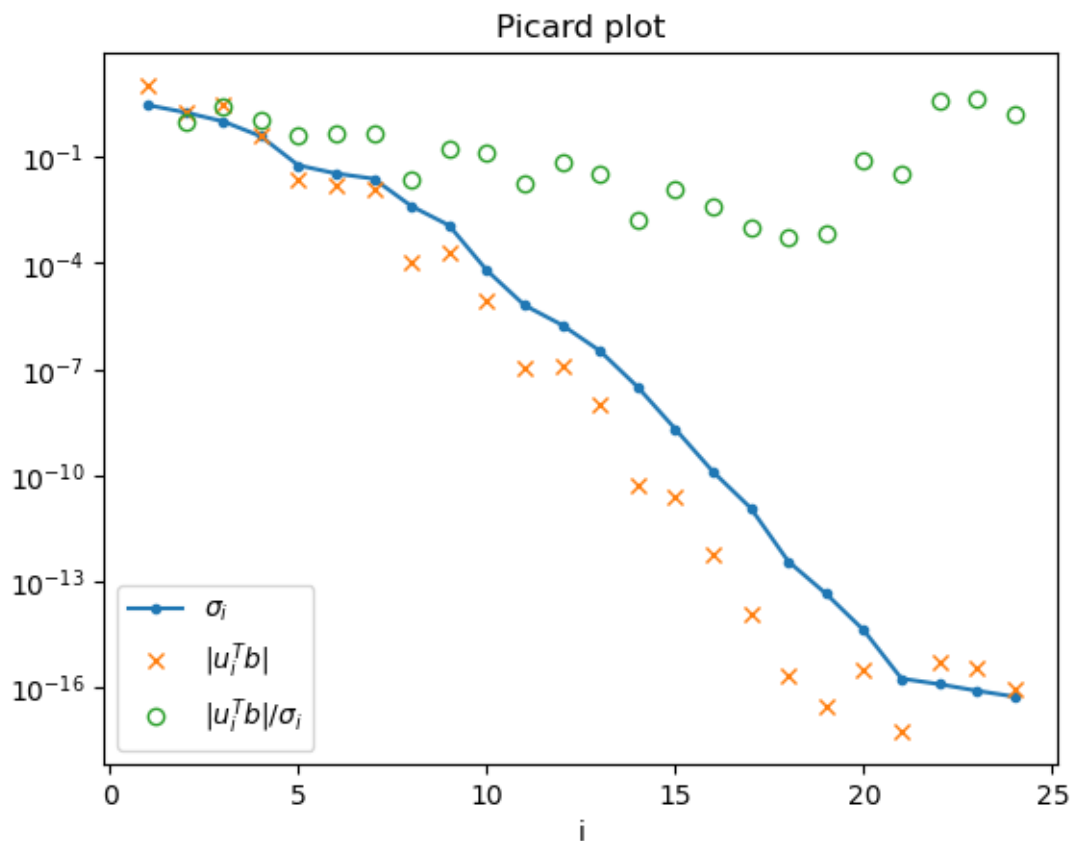


4.2 Part B

Use the function `picard` from `hw2_helper_funcs.py` to inspect the singular values and SVD coefficients. Is the Picard condition satisfied? Why or why not?

Response:

```
[31]: eta = picard(U, s, b_exact)
```



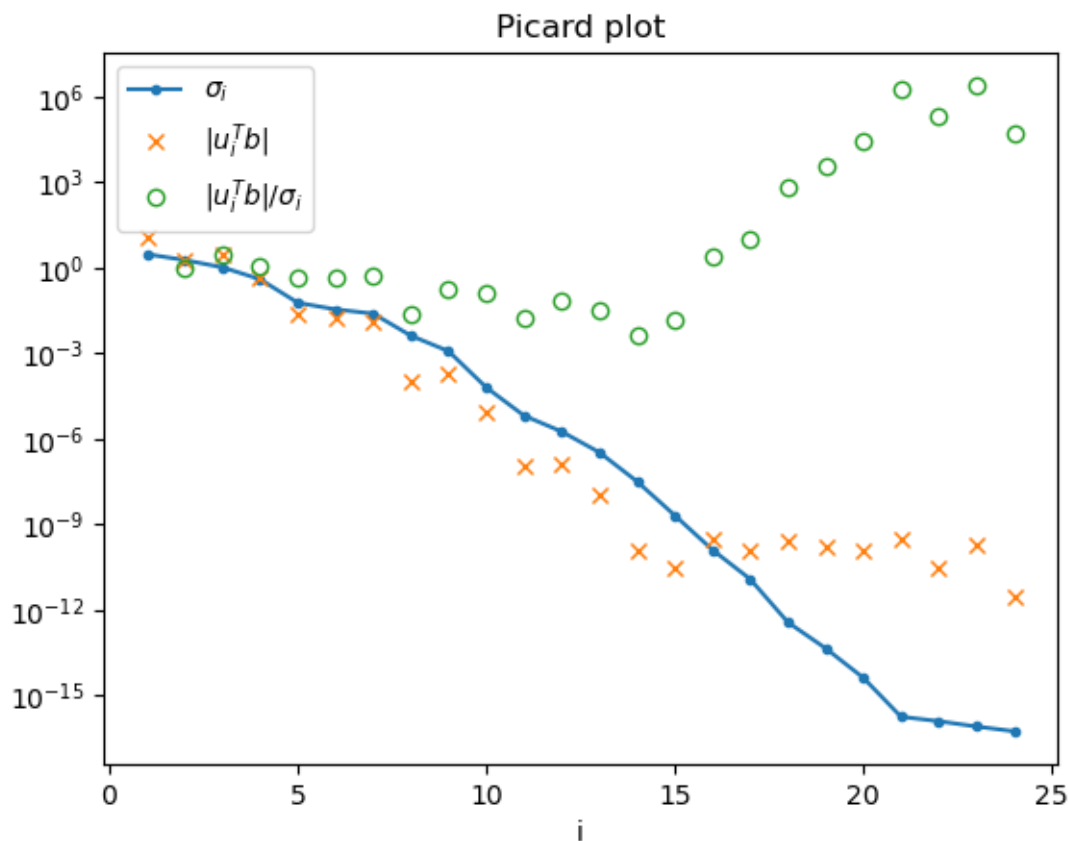
4.3 Part C

Add a small amount of noise to b_{exact} , i.e., $b = b_{\text{exact}} + e$ with $\|e\|_2/\|b_{\text{exact}}\|_2 = 10^{-10}$. Inspect the new Picard plot. What happens to the SVD coefficients $u_i^T b$ corresponding to small singular values?

Response:

```
[32]: np.random.seed(0)
      e = np.random.normal(size=len(b_exact))
      e = (e/np.linalg.norm(e))*np.linalg.norm(b_exact)*(1e-10)
      b_noisy = b_exact + e
```

```
[33]: eta = picard(U, s, b_noisy)
```



5 Part D

Compute the partial sums

$$x_k = \sum_{i=1}^k \frac{u_i^T b}{\sigma_i} v_i, \quad k = 1, \dots$$

and inspect the vectors x_k (plot them). Try to explain the behavior of these vectors.

Response:

```
[36]: ks = np.arange(1,25,3)
fig, axs = plt.subplots(2,4, figsize=(13,8))

# Iterate over each axis
for j, ax in enumerate(axs.flat):

    # Plot exact solution
    ax.plot(x_exact, label="Exact", color="k", ls="--")
    if j == 0:
        ax.legend()
```

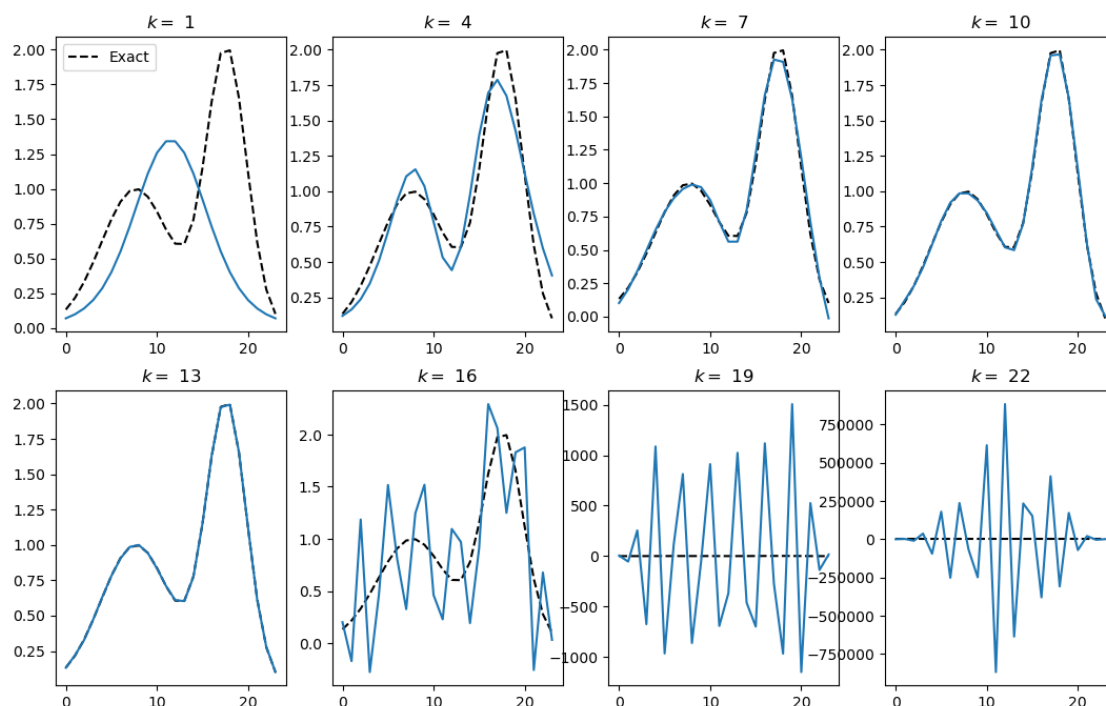
```

# Title
ax.set_title(f"$k = $ {ks[j]}")

# Compute xk
k = ks[j]
xk = np.zeros_like(x_exact)
for i in range(k):
    xk += (np.dot(U[:,i], b_noisy)/s[i])*V[:,i]

# Plot xk
ax.plot(xk)

```



6 Problem 3.7

6.1 Part A

Choose $m = 40$ and $\eta = 10^{-5}$ and generate a number of instances of white Gaussian noise with standard deviation η , by means of $\mathbf{e} = \text{eta} * \text{np.random.normal(size=m)}$. Check that the computed values of $\mathcal{E}(e)$, $\mathcal{E}(\|e\|_2^2)$, and $\mathcal{E}(\|e\|_2)$ are in accordance with the results in (3.20). Are they?

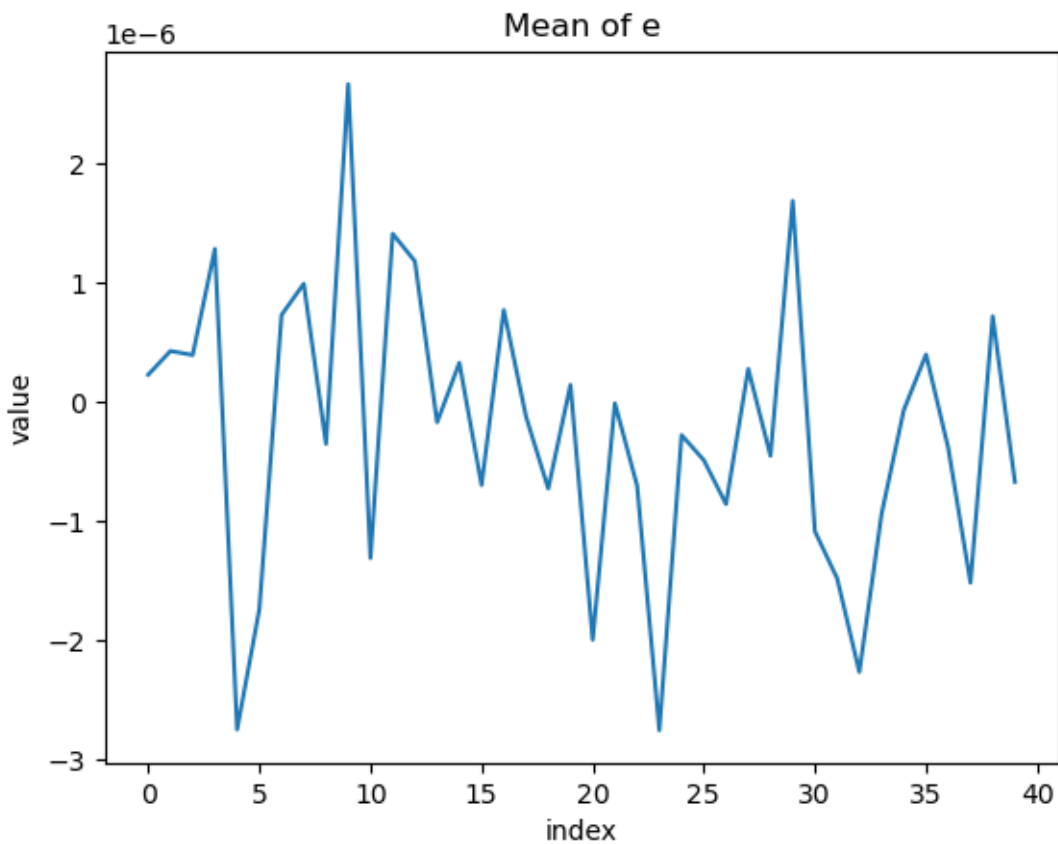
Response:

```
[50]: m = 40
      eta = 1e-5
```

```
[54]: n_instances = 100
      np.random.seed(0)
      randvecs = eta*np.random.normal(size=(n_instances,m))

      # Compute mean of e
      mean_e = np.mean(randvecs, axis=0)

      plt.plot(mean_e)
      plt.title("Mean of e")
      plt.xlabel("index")
      plt.ylabel("value")
      plt.show()
```



```
[60]: # Compute mean of || e ||_2^2
      e_norms = []
      for j in range(n_instances):
          e_norms.append(np.linalg.norm(randvecs[j,:]))
```



```
e_norms = np.asarray(e_norms)
e_norms_sq = e_norms**2
mean_e_norm = np.mean(e_norms)
mean_e_norm_sq = np.mean(e_norms_sq)

print(f"Mean of || e ||_2: {mean_e_norm:5e}")
print(f"Mean of || e ||_2^2: {mean_e_norm_sq:5e}")
```

Mean of || e ||_2: 6.179980e-05
Mean of || e ||_2^2: 3.859000e-09

```
[63]: theoretical_mean_e_norm = eta*np.sqrt(2)*gamma_func((m+1)/2)/gamma_func(m/2)
theoretical_mean_e_norm_sq = m*(eta**2)
print(f"Theoretical mean of || e ||_2: {theoretical_mean_e_norm:5e}")
print(f"Theoretical mean of || e ||_2^2: {theoretical_mean_e_norm_sq:5e}")
```

Theoretical mean of || e ||_2: 6.285154e-05
Theoretical mean of || e ||_2^2: 4.000000e-09

6.2 Part B

Set up the same shaw problem from the previous problem, with exact rhs vector b_{exact} . Add a noise vector e from part A to get a noisy rhs vector b_{noisy} . What is the relative noise level $\|e\|_2/\|b_{\text{exact}}\|_2$

Response:

```
[64]: # Setup shaw problem
A, b_exact, x_exact = shaw(m)
svdres = np.linalg.svd(A)
U, s, Vt = svdres.U, svdres.S, svdres.Vh
S = np.diag(s)
V = Vt.T
```

```
[66]: np.random.seed(0)
e = eta*np.random.normal(size=m)
noise_level = np.linalg.norm(e)/np.linalg.norm(b_exact)
print(f"Noise level: {noise_level:3e}")
```

Noise level: 4.758414e-06

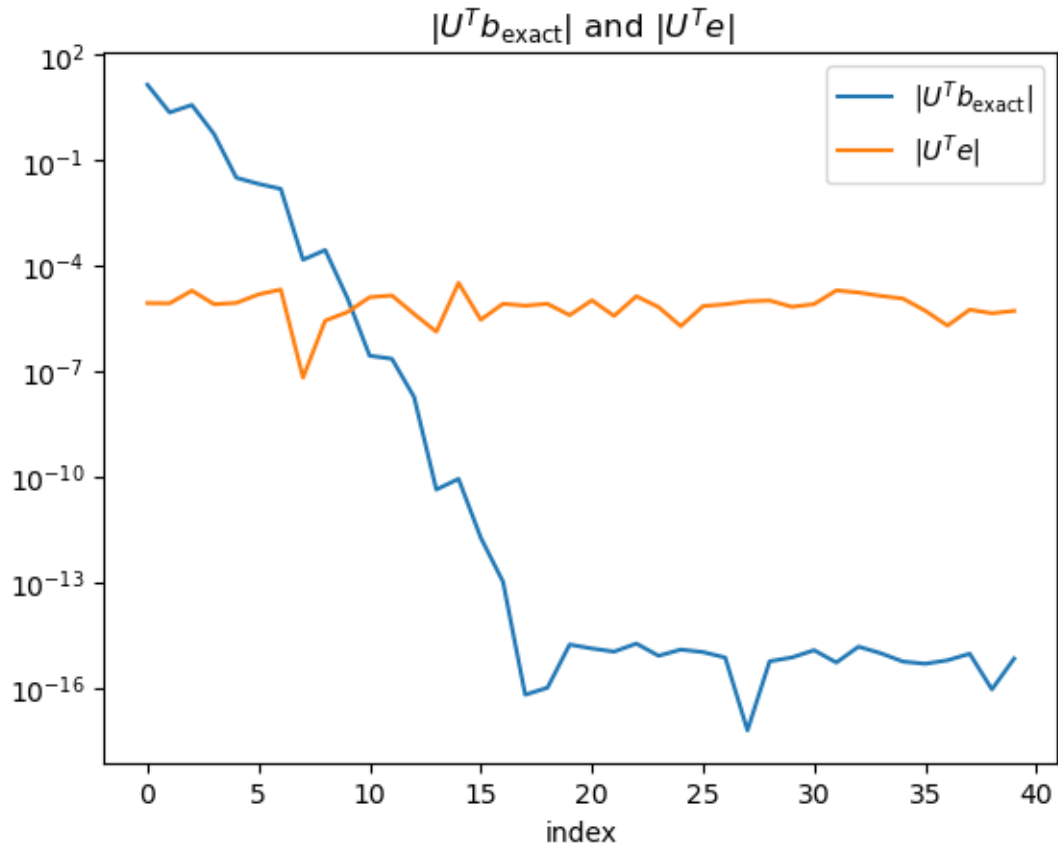
6.3 Part C

Use a semilogarithmic plot to show absolute values of the elements of $U^T b_{\text{exact}}$ and $U^T e$, and explain the behavior of these plots. In particular, explain why both graphs tend to level off in the right part of the plot, and explain the magnitude of the plateau.

Response:

```
[78]: plt.semilogy(np.abs(U.T @ b_exact), label="$|U^T b_{\\text{exact}}|$")
plt.semilogy(np.abs(U.T @ e), label="$|U^T e|$")
```

```
plt.legend()
plt.xlabel("index")
plt.title("$|U^T b_{\text{exact}}|$ and $|U^T e|$")
plt.show()
```



6.4 Part D

Show that if you want to generate a noisy rhs with a given noise level $\|e\|_2/\|b_{\text{exact}}\|_2 = \text{rnl}$, you should use the Python code:

```
e = np.random.normal(m)
e = e/np.linalg.norm(e)
e = rnl*np.linalg.norm(b_exact)*e
b = b_exact + e
```

Response: