```
import numpy as np
from scipy.linalg import hilbert
from scipy.sparse.linalg import cg
from scipy.linalg import solve
from numpy.linalg import cond
import pandas as pd

# Define the matrix dimensions to be tested
n_values = [5, 9, 20, 100]
results = []

for n in n_values:
    # Create Hilbert matrix and exact solution
    A = hilbert(n)
    x_exact = np.ones(n)
    b = A @ x_exact  # Generate b based on known solution x

    # 1. Compute the condition number
    condition_number = cond(A)

    # 2. Solve using the direct method
    x_direct = solve(A, b)
    error_direct = np.linalg.norm(x_exact - x_direct)

    # 3. Solve using Preconditioned Gradient Descent (PG) method with
iteration count
    def preconditioned_gradient_descent(A, b, M, x0=None, tol=1e-7,
max_iterations=100000):
        n = len(b)
        x = np.zeros_like(b) if x0 is None else x0
        iteration_count = 0  # Initialize iteration counter
        r = b - A @ x  # Initial residual
        while iteration_count < max_iterations and np.linalg.norm(r) >
tol:
            z = M @ r  # Apply preconditioner
            alpha = (r @ z) / (z @ (A @ z))  # Compute step size
            x += alpha * z  # Update solution
            r -= alpha * (A @ z)  # Update residual
            iteration_count += 1  # Increment iteration count
        return x, iteration_count

    # Diagonal preconditioner for PG
    M_pg = np.diag(1 / np.diag(A))
    x_pg, pg_iterations = preconditioned_gradient_descent(A, b, M_pg)
    error_pg = np.linalg.norm(x_exact - x_pg)

    # 4. Solve using PCG method with a diagonal preconditioner and
custom iteration counter
    M_pcg = np.diag(1 / np.diag(A))  # Preconditioner matrix as the
inverse of the diagonal entries of A
```

```
    # Custom iteration counter using a mutable list
    iteration_count = [0]
    def iteration_callback(xk):
        iteration_count[0] += 1

    # Use CG with preconditioning and capture the iteration
information
    x_pcg, pcg_info = cg(A, b, M=M_pcg, tol=1e-10, maxiter=10000,
callback=iteration_callback)
    error_pcg = np.linalg.norm(x_exact - x_pcg)

    # Set the PCG iteration count based on convergence information
    pcg_iterations = iteration_count[0] if pcg_info == 0 else "Reached
Maximum Iterations"

    # Save results
    results.append({
        'n': n,
        'K(A)': f"{condition_number:.2e}",
        'Direct Error': f"{error_direct:.2e}",
        'PG Error': f"{error_pg:.2e}",
        'PG Iter': pg_iterations,
        'PCG Error': f"{error_pcg:.2e}",
        'PCG Iter': pcg_iterations
    })

# Display results in a nicely formatted table
df = pd.DataFrame(results)
df = df.rename(columns={
    'n': 'n', 'K(A)': 'K(A)', 'Direct Error': 'Direct Error', 'PG
Error': 'PG Error',
    'PG Iter': 'PG Iter', 'PCG Error': 'PCG Error', 'PCG Iter': 'PCG
Iter'
})
df.index = range(1, len(df) + 1)
styled_df = df.style.set_table_styles(
    [{'selector': 'th', 'props': [('font-weight', 'bold')]}]
).set_caption("Results for Solving Hilbert Matrix Linear Systems")

# For Jupyter Notebook, display styled DataFrame
styled_df
```

```
C:\Users\jhyang\AppData\Local\Temp\ipykernel_17396\3310942156.py:46:
DeprecationWarning: 'scipy.sparse.linalg.cg' keyword argument `tol` is
deprecated in favor of `rtol` and will be removed in SciPy v1.14.0.
Until then, if set, it will override `rtol`.
  x_pcg, pcg_info = cg(A, b, M=M_pcg, tol=1e-10, maxiter=10000,
callback=iteration_callback)
C:\Users\jhyang\AppData\Local\Temp\ipykernel_17396\3310942156.py:15:
```

```
LinAlgWarning: Ill-conditioned matrix (rcond=5.62878e-20): result may
not be accurate.
  x_direct = solve(A, b)
C:\Users\jhyang\AppData\Local\Temp\ipykernel_17396\3310942156.py:15:
LinAlgWarning: Ill-conditioned matrix (rcond=3.29506e-21): result may
not be accurate.
  x_direct = solve(A, b)

<pandas.io.formats.style.Styler at 0x233e9bb2bd0>
```