```python
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, atol=1e-10, maxiter=100000,
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 the results as a DataFrame
df = pd.DataFrame(results)

# Rename columns for better readability
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'
})

# Reset the index to start from 1 for better readability
df.index = range(1, len(df) + 1)

# Print the DataFrame as plain text
print(df.to_string(index=True))
```

```
C:\Users\jhyang\AppData\Local\Temp\ipykernel_16576\1634977234.py:15:
LinAlgWarning: Ill-conditioned matrix (rcond=2.93284e-20): result may
not be accurate.
```

```
  x_direct = solve(A, b)
C:\Users\jhyang\AppData\Local\Temp\ipykernel_16576\1634977234.py:15:
LinAlgWarning: Ill-conditioned matrix (rcond=8.9205e-21): result may
not be accurate.
  x_direct = solve(A, b)
```

	n	K(A)	Direct Error	PG Error	PG Iter	PCG Error	PCG Iter
1	5	4.77e+05	4.03e-12	4.36e-03	6823	4.22e-02	3
2	9	4.93e+11	5.26e-05	4.42e-03	12489	2.79e-02	4
3	20	1.16e+18	1.60e+02	6.71e-03	13419	3.30e-02	5
4	100	1.08e+19	2.95e+03	6.61e-03	61033	1.32e-01	6