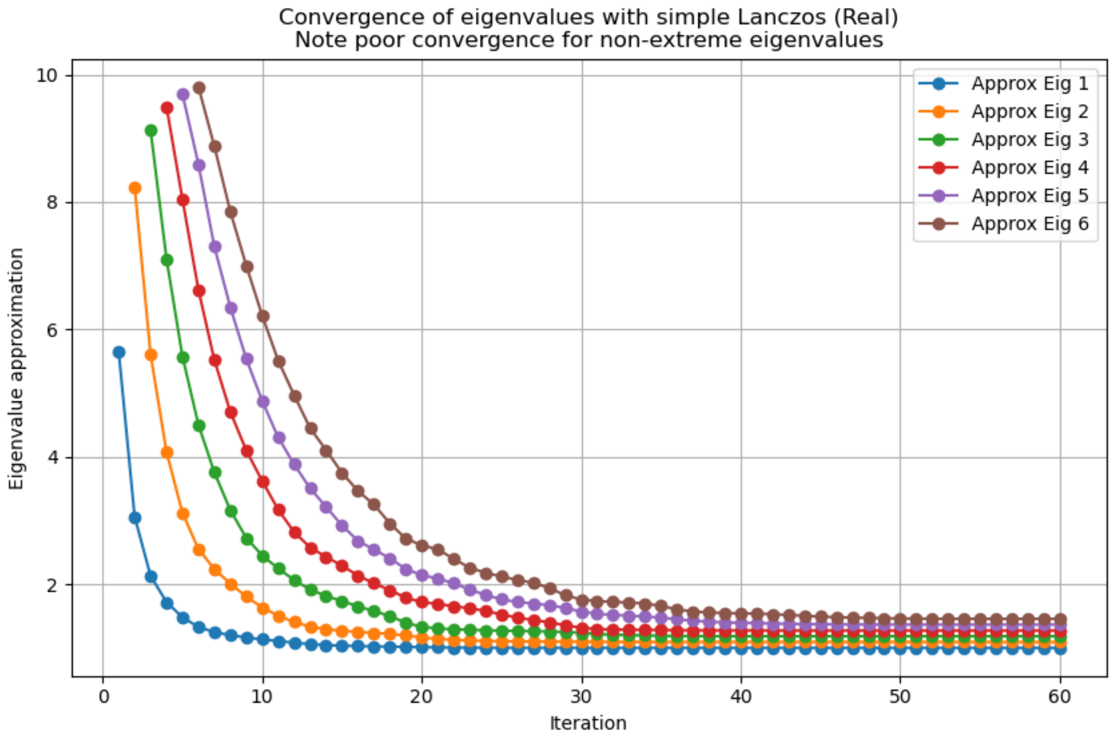
lanczos

December 8, 2024

simple lanczos

[2]:
```python
import numpy as np
from scipy.linalg import eigh_tridiagonal
import matplotlib.pyplot as plt

# Parameters
n = 100
max_iter = 60
num_eigs_to_track = 6

# Construct a real symmetric (Hermitian) matrix
true_eigvals = np.linspace(1.0, 10.0, n)
H = np.diag(true_eigvals)  # This is already Hermitian and real

# Choose a real initial vector
q = np.random.rand(n)
q = q / np.linalg.norm(q)

q_list = [np.zeros(n), q]
alpha_list = []
beta_list = []

eigs_history = []

for i in range(1, max_iter + 1):
    # Lanczos iteration
    x = H @ q_list[i]
    alpha = np.vdot(q_list[i], x)
    # alpha should be real:
    alpha = alpha.real
    alpha_list.append(alpha)

    if i == 1:
        x = x - alpha * q_list[i]
    else:
        x = x - alpha * q_list[i] - beta_list[i-2]*q_list[i-1]
```

```python
    if i < max_iter:
        beta = np.linalg.norm(x)
        beta_list.append(beta)
        if beta != 0:
            q_list.append(x / beta)
        else:
            break

    # Construct T_i
    alphas = np.array(alpha_list[:i], dtype=float)
    betas = np.array(beta_list[:i-1], dtype=float)

    # eigh_tridiagonal requires real arrays
    approx_eigvals = eigh_tridiagonal(alphas, betas)[0]  # [0] to get just eigenvalues
    approx_eigvals = np.sort(approx_eigvals.real)  # ensure they're real and sorted

    k = min(num_eigs_to_track, len(approx_eigvals))
    tracked_eigs = approx_eigvals[:k]
    if k < num_eigs_to_track:
        padded = np.full((num_eigs_to_track,), np.nan, dtype=tracked_eigs.dtype)
        padded[:k] = tracked_eigs
        tracked_eigs = padded
    eigs_history.append(tracked_eigs)

eigs_history = np.array(eigs_history)

# Plot
plt.figure(figsize=(10,6))
iterations = np.arange(1, max_iter+1)
for j in range(num_eigs_to_track):
    plt.plot(iterations, eigs_history[:, j], marker='o', label=f'Approx Eig {j+1}')

plt.xlabel('Iteration')
plt.ylabel('Eigenvalue approximation')
plt.title('Convergence of eigenvalues with simple Lanczos (Real)\nNote poor convergence for non-extreme eigenvalues')
plt.grid(True)
plt.legend()
plt.show()
```

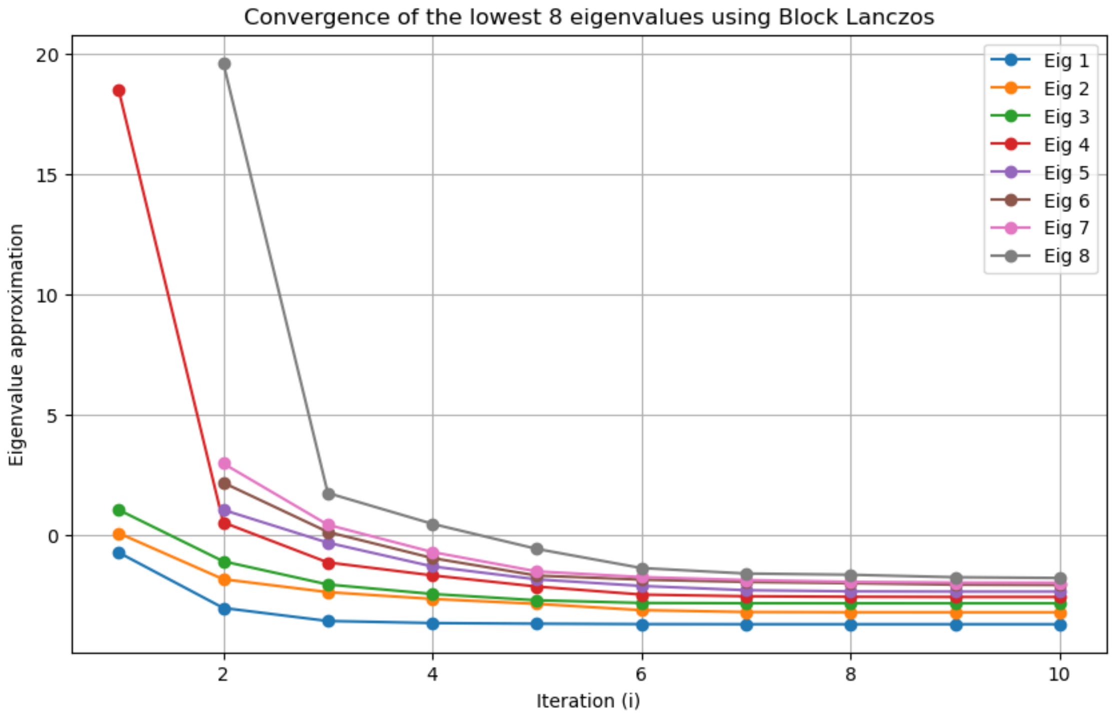


block lanczos

[3]:
```python
import numpy as np
from scipy.linalg import qr, eigh
import matplotlib.pyplot as plt

def block_lanczos_iteration(H, Q_i, Q_im1, B_im1):
    """
    Perform one step of the block Lanczos iteration.
    H: Hermitian matrix (n x n)
    Q_i: Current block vector (n x b)
    Q_im1: Previous block vector (n x b) or None if not applicable
    B_im1: Previous B block (b x b) or None if not applicable

    Returns:
    A_i, B_i, Q_ip1
    """
    W = H @ Q_i
    A_i = Q_i.conj().T @ W
    W = W - Q_i @ A_i
    if Q_im1 is not None and B_im1 is not None:
        W = W - Q_im1 @ B_im1.conj().T
    Q_ip1, B_i = qr(W, mode='economic')
    return A_i, B_i, Q_ip1
```

```
# Parameters
n = 40
block_size = 4
max_iter = 10  # number of block iterations
num_eigs_to_track = 8 # Number of eigenvalues to track

# Construct a Hermitian matrix
H = np.random.rand(n, n) + 1j * np.random.rand(n, n)
H = (H + H.conj().T) / 2

# Initial block Q_1
Q_init = np.random.rand(n, block_size) + 1j * np.random.rand(n, block_size)
Q_init, _ = np.linalg.qr(Q_init)  # orthonormalize

Q_blocks = [Q_init]
A_blocks = []
B_blocks = []

lowest_eigs_history = []

for i in range(max_iter):
    if i == 0:
        Q_im1 = None
        B_im1 = None
    else:
        Q_im1 = Q_blocks[i-1]
        B_im1 = B_blocks[i-1]

    A_i, B_i, Q_ip1 = block_lanczos_iteration(H, Q_blocks[i], Q_im1, B_im1)
    A_blocks.append(A_i)
    B_blocks.append(B_i)
    if i < max_iter - 1:
        Q_blocks.append(Q_ip1)

    # Construct the block tridiagonal matrix T_i
    current_dim = (i+1)*block_size
    T_i = np.zeros((current_dim, current_dim), dtype=H.dtype)

    # Fill main diagonal blocks
    for k in range(i+1):
        T_i[k*block_size:(k+1)*block_size, k*block_size:(k+1)*block_size] =
 ↪A_blocks[k]

    # Fill off-diagonal blocks
    for k in range(i):
```

```
        T_i[k*block_size:(k+1)*block_size, (k+1)*block_size:(k+2)*block_size] = B_blocks[k]
        T_i[(k+1)*block_size:(k+2)*block_size, k*block_size:(k+1)*block_size] = B_blocks[k].conj().T

    # Compute eigenvalues of T_i
    eigvals = eigh(T_i, eigvals_only=True)
    eigvals = np.sort(eigvals)

    # Determine how many eigenvalues we can track at this iteration
    # (In the first few iterations, the dimension might be smaller than num_eigs_to_track)
    num_to_take = min(num_eigs_to_track, len(eigvals))
    lowest_eigs_history.append(eigvals[:num_to_take])

# Convert to array for plotting
# Note: If we always took 'num_eigs_to_track' eigenvalues, arrays align.
# If not, we can pad them. Here we assume max_iter is large enough that
# eventually num_eigs_to_track can be extracted.
max_len = max(len(e) for e in lowest_eigs_history)
for i, arr in enumerate(lowest_eigs_history):
    if len(arr) < max_len:
        # Pad with NaNs for plotting
        padded = np.full((max_len,), np.nan, dtype=arr.dtype)
        padded[:len(arr)] = arr
        lowest_eigs_history[i] = padded

lowest_eigs_history = np.array(lowest_eigs_history)  # shape: (max_iter, max_len)

# Plot the convergence of multiple eigenvalues
plt.figure(figsize=(10,6))
for j in range(max_len):
    plt.plot(range(1, max_iter+1), np.real(lowest_eigs_history[:, j]),
             marker='o', label=f'Eig {j+1}')

plt.xlabel('Iteration (i)')
plt.ylabel('Eigenvalue approximation')
plt.title(f'Convergence of the lowest {num_eigs_to_track} eigenvalues using Block Lanczos')
plt.grid(True)
plt.legend()
plt.show()
```



Convergence of the lowest 8 eigenvalues using Block Lanczos
Eigenvalue approximation
Iteration (i)
0
5
10
15
20
2
4
6
8
10
Eig 1
Eig 2
Eig 3
Eig 4
Eig 5
Eig 6
Eig 7
Eig 8