Problem 1: Where do you belong, Krylov?

Solution

a

$$Aq^i = \frac{q^{i+1}}{\|q^{i+1}\|} + \sum_k^i \langle Aq^i, q^k \rangle q^k \in \mathcal{K}_{i+1}$$

b

We firstly note that $q^1 = \frac{b}{\|b\|}$, and so we can note that it is a scaled version of b. Take this to be the first element of a Krylov subspace $\mathcal{K}^1 = q^1$. Furthermore, note that the definition of q^2 (or more generally, q^{i+1}) is equivalent to multiplying Aq^i and subtracting the projection of the prior q's. Critically, because the subspace is defined as the *span* of a set of vectors, the resulting set of orthonormal vectors define an equivalent Krylov subspace – they are, in fact, an orthonormal basis of this subspace.

So, we can say:

$$\mathcal{K}^i = \text{span}(\left\{\frac{b}{\|b\|}, Aq_1, \ldots, Aq_{i-1}\right\})$$

$$\|$$

$$\begin{bmatrix} Aq_1 & | & \ldots & | & Aq_n \end{bmatrix} = \begin{bmatrix} Qh_1 & | & \ldots & | & Qh_n \end{bmatrix}$$
$$\implies Aq_1 = Qh_1$$
$$Aq_1 \in \mathcal{K}_2 \implies Qh_1 \in \mathcal{K}_2$$
$$Aq_1 = \begin{bmatrix} * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \; Aq_2 = \begin{bmatrix} * \\ * \\ * \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \; \ldots Aq_n = \begin{bmatrix} * \\ * \\ * \\ * \\ \vdots \\ * \end{bmatrix}$$

c

See the code section at the end for the implementation. The result is:

```
Did arnoldi work:
True
Q:
[[0.58 0.31 -0.75]
        [-0.58 0.81 -0.11]
        [0.58 0.50 0.65]]
H:
[[0.33 3.09 0.00]
        [3.09 7.56 0.95]
        [0.00 0.95 -0.90]]
Wow its upper Hessenberg, whoda thunk it
Wow its lower Hessenberg, whoda thunk it
Wow it's tridiagonal, its cuz the matrix A is Hermitian (or Symmetric cuz its real)
```

Problem 2: Ask Gram, Schmidt, or Givens for QR

Solution

a

Taken from "Linear Algebra Done Right" by Sheldon Axler.

Suppose $v_1, \ldots, v_m$ is a lin. ind. list of vectors in V. Let $e_1 = \frac{v_1}{\|v_1\|}$. For $j = 2, \ldots, m$, define e_j inductively by:

$$e_j = \frac{v_j - \langle v_j, e_1 \rangle e_1 - \cdots - \langle v_j, e_{j-1} \rangle e_{j-1}}{\|v_j - \langle v_j, e_1 \rangle e_1 - \cdots - \langle v_j, e_{j-1} \rangle e_{j-1}\|}$$

(i.e. the Gram-Schmidt process definition). Then, $e_1, \ldots, e_m$ is an orthonormal list of vectors in V s.t.:

$$\operatorname{span}(v_1, \ldots, v_j) = \operatorname{span}(e_1, \ldots, e_j) \forall j = 1, \ldots, m$$

Note that for $j = 1$, $\operatorname{span}(v_1) = \operatorname{span}(e_1)$ because v_1 is a positive mulitple of e_1.

Suppose $1 < j < m$ and it has been verified that:

$$\operatorname{span}(v_1, \ldots, v_{j-1}) = \operatorname{span}(e_1, \ldots, e_{j-1})$$

Note that $v_j \notin \operatorname{span}(v_1, \ldots, v_{j-1})$ because $v_1 \ldots, v_m$ are linearly independent. Thus, $v_j \notin \operatorname{span}(e_1, \ldots, e_{j-1})$. As such, we are not dividing by zero in the definition of e_j. We can also see that $\|e_j\| = 1$ by its definition.

Let $k \in [1, j)$. Then:

$$\begin{aligned}
\langle e_j, e_k \rangle &= \left\langle \frac{v_j - \langle v_j, e_1 \rangle e_1 - \ldots - \langle v_j, e_{j-1} \rangle e_{j-1}}{v_j - \langle v_j, e_1 \rangle e_1 - \ldots - \langle v_j, e_{j-1} \rangle e_{j-1}}, e_k \right\rangle \\
&= \frac{\langle v_j, e_k \rangle - \langle v_j, e_k \rangle}{v_j - \langle v_j, e_1 \rangle e_1 - \ldots - \langle v_j, e_{j-1} \rangle e_{j-1}} \\
&= 0
\end{aligned}$$

Thus $e_1, \ldots, e_j$ is an orthonormal list.

From the definition of e_j, we see that $v_j \in \operatorname{span}(e_1, \ldots, e_j)$. Combined with the equivalency of the spans provided above, we know:

$$\operatorname{span}(v_1, \ldots, v_j) \subset \operatorname{span}(e_1, \ldots, e_j)$$

Both these lists are lin. ind., thus both subspaces have dimension j and are equal. □

b

The inner loop requires:

$$(4m-1)(1-1)+(4m-1)+(2-1)+\ldots+(4m-1)(n-1) = (4m-1)(0+1+\ldots+n-1) = (4m-1)(\frac{1}{2}n(n+1)-1)$$

The outer portion requires $(4m-1) * n$ operations. Summing:

$$= (4m+1)(\frac{1}{2}n(n+1)-1) + (4m+1)(n) = (4m+1)(\frac{1}{2}n(n)-1+n)$$

Simplifying and discarding lower order terms we find:

$$= 2mn^2$$

Algorithm 1 Classical Gram-Schmidt, annotated with FLOP counts

for $j = 1, \ldots, n$ **do**
  $v \leftarrow a^j$
  for $i = 1, \ldots, j-1$ **do**
    $R_{ij} \leftarrow \langle v, q^i \rangle$ $\{m$ multiplications, $(m-1)$ additions $\implies 2m-1\}$
    $v \leftarrow v - R_{ij} q^i$ $\{m$ multiplications, m subtractions $\implies 2m\}$
  end for{Total cost is }
  $q^j \leftarrow \frac{v}{\|v\|}$ $\{m$ multiplications, $m-1$ additions, 1 for sqrt $\implies 2m\}$
  $R_{jj} \leftarrow \langle a^j, q^j \rangle$ $\{m$ multiplications, $(m-1)$ additions $\implies 2m-1\}$
end for
$Q = (q^1 | \ldots | q^n)$

c

An upper Hessenberg matrix of size $n \times n$ will require $n-1$ operations to zero out the non-zero elements below the diagonal, which will result in a QR decomposition. Givens matrices take the form:

$$G_n = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & \vdots \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

The location of the c and s terms correspond to zeroing element i and j; that is, to zero element $A_{i,j}$, we set $s = G_{ji} = -Gij$ and $G_{k,k} = c$ for $k = i, j$ (and 1 for $k \neq i, j$). The resulting algorithm is as follows:

Algorithm 2 QR Factorization of an Upper Hessenberg Matrix using Givens rotations

Input: $A^1 \in \mathbb{R}^{n \times n}$, $A_{ij} = 0 \forall i > j+1$ {This could be a parfor}
for $j = 1, \ldots, n-1$ **do**
  $i \leftarrow j+1$
  $r^j \leftarrow \sqrt{\left[A^j_{jj}\right]^2 + \left[A^j_{ij}\right]^2}$
  $c \leftarrow A^j_{jj}/r$
  $s \leftarrow -A^j_{ij}/r$
  $G^j_{ji} \leftarrow s$
  $G^j_{ij} \leftarrow -s$
  $G^j_{ii} \leftarrow c$
  $G^j_{jj} \leftarrow c$
  $A^{j+1} \leftarrow G^j A^j$
end for
$Q = \Pi_{j=1}^{n-1} G^j$
$R = A^{n-1}$

Code

Problem 1

```python
#!/usr/bin/env python3
import numpy as np

float_formatter = "{:.2f}".format

np.set_printoptions(formatter={"float_kind": float_formatter})


def is_upper_hessenberg(A: np.ndarray) -> bool:
    return np.isclose(np.tril(A, k=-2), 0.0, atol=1e-3).all()


def is_lower_hessenberg(A: np.ndarray) -> bool:
    return np.isclose(np.triu(A, k=2), 0.0, atol=1e-3).all()


# Adapted from: https://relate.cs.illinois.edu/course/cs450-s18/file-version/587
    fb0505883e0622b9b144b532b5e9a4e7a3684/demos/upload/04-eigenvalues/Arnoldi%20iteration.html
def arnoldi_iteration(A, b):
    n = A.shape[0]
    H = np.zeros((n, n))
    Q = np.zeros((n, n))
    # Normalize the input vector
    Q[:, 0] = b / np.linalg.norm(b)  # Use it as the first Krylov vector
    for k in range(n):
        u = A @ Q[:, k]
        for j in range(k + 1):
            qj = Q[:, j]
            H[j, k] = qj @ u
            u -= H[j, k] * qj
        if k + 1 < n:
            H[k + 1, k] = np.linalg.norm(u)
            Q[:, k + 1] = u / H[k + 1, k]

    return Q, H


if __name__ == "__main__":
    A = np.array([[1, 2, 3], [2, 2, 4], [3, 4, 4]])
    b = np.array([1, -1, 1])

    q, h = arnoldi_iteration(A, b)
    print("Did arnoldi work:")
    print(np.isclose((q.T @ A @ q - h) / np.linalg.norm(A), 0.0, atol=1e-3).all())

    print("Q:")
    print(q)
    print("H:")
    print(h)
    uh = is_upper_hessenberg(h)
    lh = is_lower_hessenberg(h)
    if uh:
        print("Wow its upper Hessenberg, whoda thunk it")
    if lh:
        print("Wow its lower Hessenberg, whoda thunk it")
    if lh and uh:
        print(
            "Wow it's tridiagonal, its cuz the matrix A is Hermitian (or Symmetric cuz its real)"
        )
```