(1) Show that the Hilbert matrix is positive definite.
Proof:

Suppose we have the $n \times n$ Hilbert matrix

$$H_{ij} = \frac{1}{i+j-1}, \quad i,j = 1,\dots,n.$$

Then, to show that H is positive definite, we need to show that $x^*Hx > 0$ for all $x \neq 0$ in $\mathbb{C}^n$. So, suppose we have any nonzero $x \in \mathbb{C}^n$. Then,

$$\begin{aligned}
x^*Hx = \sum_{i=1}^{n}\left(\bar{x}_i \sum_{j=1}^{n} x_j H_{ij}\right) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \bar{x}_i x_j \frac{1}{i+j-1} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \bar{x}_i x_j \int_0^1 t^{i+j-2}\mathrm{d}t \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \int_0^1 (\bar{x}_i t^{i-1})(x_j t^{j-1})\mathrm{d}t \\
&= \int_0^1 \sum_{i=1}^{n}\sum_{j=1}^{n} (\bar{x}_i t^{i-1})(x_j t^{j-1})\mathrm{d}t \\
&= \int_0^1 \left(\sum_{i=1}^{n} \bar{x}_i t^{i-1}\right)\left(\sum_{j=1}^{n} x_j t^{j-1}\right)\mathrm{d}t.
\end{aligned}$$

Now if we let $\alpha = \sum_{j=1}^{n} x_j t^{j-1}$, our equation becomes

$$\begin{aligned}
x^*Hx &= \int_0^1 \left(\sum_{i=1}^{n} \bar{x}_i t^{i-1}\right)\left(\sum_{j=1}^{n} x_j t^{j-1}\right)\mathrm{d}t \\
&= \int_0^1 \bar{\alpha}\alpha \mathrm{d}t \\
&= \int_0^1 |\alpha|^2 \,\mathrm{d}t \\
&> 0.
\end{aligned}$$

So, our Hilbert matrix must be positive definite. □

(2) Using my power iteration code (attached in section: Code Used), I generated the table below of the largest eigenvalues for the first 10 square Hilbert matrices.

n	Eigenvalue	Eigenvector
1	1.00	$[1.0]^T$
2	1.27	$[0.88, 0.47]^T$
3	1.41	$[0.83, 0.46, 0.32]^T$
4	1.50	$[0.79, 0.45, 0.32, 0.25]^T$
5	1.57	$[0.77, 0.45, 0.32, 0.25, 0.21]^T$
6	1.62	$[0.75, 0.44, 0.32, 0.25, 0.21, 0.18]$
7	1.66	$[0.73, 0.44, 0.32, 0.25, 0.21, 0.18, 0.16]^T$
8	1.70	$[0.72, 0.43, 0.32, 0.26, 0.21, 0.18, 0.16, 0.15]^T$
9	1.73	$[0.71, 0.43, 0.32, 0.26, 0.21, 0.19, 0.16, 0.15, 0.13]^T$
10	1.75	$[0.70, 0.43, 0.32, 0.26, 0.22, 0.19, 0.16, 0.15, 0.13, 0.12]^T$

(3) To find the smallest eigenvalues using a power iteration, I just change the matrix-vector multiplication to a backslash as to solve $Ax = b$ instead of computing $x = Ab$. This change has the effect of making the smallest eigenvalue of A dominate the power iteration. For $n = 16$, I obtained the eigenvalue:

$$\lambda_{\min} = -2.765686556840889 \cdot 10^{-18}$$

which is not even positive even though we know Hilbert matrices are symmetric positive definite. So my naive Julia code is definitely not converging properly under double precision. Putting my Julia code in quad precision yields the eigenvalue

$$\lambda_{\min} = 9.197419820719313 \cdot 10^{-23}.$$

which is consistent with Mathematica's eigenvalue. Hence, we can see that my computed double-precision eigenvalue is off by over 5 orders of magnitude. However, this result is still consistent with our error estimate

$$\min_{\lambda \in \sigma(A)} |\mu - \lambda| \leq \|E\|_2$$

because $\|E\|_2$ should be on the order of 10^{-16} which is the best we can do in double precision.

(4) Assume that a real matrix A has eigenvalues $\lambda_1 = -\lambda_2$ and $|\lambda_1| = |\lambda_2| > |\lambda_3| \geq |\lambda_n|$.

Without loss of generality, assume $\lambda_1 > 0$ which makes $\lambda_2 < 0$. Now, note that because A is symmetric, eigenvectors corresponding to different eigenvalues of A are orthogonal. Then, to find the eigenvectors corresponding to λ_1 and λ_2, run the standard power iteration to find some normalized vector, $\vec{v}_0$, in the span of $\vec{\lambda}_1$ and $\vec{\lambda}_2$ where $\vec{\lambda}_1$ and $\vec{\lambda}_2$ are eigenvectors corresponding to λ_1 and λ_2 respectively. Then, because $\vec{\lambda}_1$ and $\vec{\lambda}_2$ are orthogonal, we can uniquely decompose $\vec{v}_0$ as

$$\vec{v}_0 = a\vec{\lambda}_1 + b\vec{\lambda}_2$$

for some constants a and b. Then, one more power iteration on $\vec{v}_0$ will yield

$$\vec{v}_1 = a\vec{\lambda}_1 - b\vec{\lambda}_2$$

because $\lambda_2 < 0$ and the eigenvectors are orthogonal. Then, we can simply find an eigenvector corresponding to λ_1 as

$$\vec{v}_0 + \vec{v}_1 = 2a\vec{\lambda}_1$$

and an eigenvector corresponding to λ_2 as

$$\vec{v}_0 - \vec{v}_1 = 2b\vec{\lambda}_2.$$

(5) A real symmetric matrix A has an eigenvalue 1 of multiplicity 8; the rest of the eigenvalues are ≤ 0.1 in absolute value.

We can find an orthogonal basis for the 8-dimensional eigenspace corresponding to the dominant eigenvalue by combining the standard power iteration with the Gram-Schmidt process. To do so:

(i) Generate 8, random vectors.

(ii) Apply power iteration to each vector until convergence. The resulting vectors will be in the span of the dominant eigenvectors.

(iii) Apply Gram-Schmidt to the set of generated vectors.

(iv) You're all done!

To estimate how long it will take to find this basis for an $n \times n$ matrix, we need to figure out how long it will take each power iteration to converge to double-precision. If each subdominant eigenvalue has a magnitude less than or equal to 0.1, we should expect each power iteration to converge in 16 to 17 iterations because $|\lambda_{\text{next}}/\lambda_1| \leq 0.1$. Then, we need to run power iteration for each of the 8 vectors for a total of 136 iterations.

Code Used

Note that some characters are missing because LaTeX can't display some of the unicode characters in my code.

```
#=
# Power iteration method for finding eigenvalues
#
# Author: Caleb Jacobs
# Date last modified: 01-02-2022
=#

using LinearAlgebra
using DoubleFloats

function makeHilbert(n, quad = false)
    # Check if we want quad-precision matrix
    if !quad
        A = [Float64(1) / (i + j - 1) for i = 1 : n, j = 1 : n]
    else
        A = [Double64(1) / (i + j - 1) for i = 1 : n, j = 1 : n]
    end

    return A
end

function powIt(A, maxIts)
    q = z = rand(size(A, 1))    # Initialize eigenvectors
     = 0                       # Initialize eigenvalue
    for k = 1 : maxIts
        z = A * q               # Compute next eigenvector guess
        q = z / norm(z)         # Normalize eigenvector
         = q' * A * q           # Compute next eigenvalue guess
    end

    return  (, q)
end

function invIt(A, maxIts)
    q = z = ones(Float64, size(A, 1))     # Initialize eigenvectors
     = 0                                  # Initialize eigenvalue

    # Perform inverse iteration to find smallest eigenvalue
    for k = 1 : maxIts
        z = A \ q               # Compute next eigenvector guess
        q = z / norm(z)         # Normalize eigenvector
         = dot(q, A, q)         # Compute next eigenvalue guess
    end

    return
```

```
end

function prob2()
    a = 1;          # Lower size of matrix
    b = 10;         # Upper size of matrix
    maxIts = 100;   # Number of iterations for powIt

    data = Vector{Tuple{Float64, Vector{Float64}}}(undef, b - a + 1)

    n = [a : b...]  #

    for i   1 : b - a + 1
        A = makeHilbert(n[i])

        data[i] = powIt(A, maxIts)
    end

    for i   data
        print(round.(i[1]; digits = 2))
        println(round.(i[2]; digits = 2))
    end
    return(data)
end

function prob3()
    AD = makeHilbert(16)        # Double precision hilbert
    AQ = makeHilbert(16, true)  # Quad precision hilbert

    doubVal = invIt(AD, 1000)   # Double precision
    quadVal = invIt(AQ, 1000)   # Quad precision

    return (doubVal, quadVal)
end
```