+3 for presentation

CS375 HW11

Ryan Scherbarth

November 2024

1. **Since we spent a week or two learning to solve nonlinear equations, one might assume that finding the characteristic polynomial of a matrix and then using one of our root-finding methods would be a good appraoch for finding the eigenvalues. In this problem, we illustrate one reason why this is generally a bad idea. Assume that**

$$A = \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix}$$

(a) **What is the characteristic polynomial of A?**

$$\begin{aligned} det(A - \lambda I) &= 0 \\ &= \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \\ &= \begin{bmatrix} 1-\lambda & \epsilon \\ \epsilon & 1-\lambda \end{bmatrix} \end{aligned}$$

We find the determinant: $det(A - \lambda I) = (1-\lambda)(1-\lambda) - \epsilon * \epsilon = (1-\lambda)^2 - \epsilon^2$.

$$\begin{aligned} det(A - \lambda I) &= (1-\lambda)^2 - \epsilon^2 \\ &= 1 - 2\lambda + \lambda^2 - \epsilon^2 \\ &= \lambda^2 - 2\lambda + (1-\epsilon)^2 \end{aligned}$$

this first line is good

So the characteristic polynomial is $\lambda^2 - 2\lambda + (1-\epsilon)^2 = 0$.

(b) **Calculate the eigenvalues and eigenvectors of A (by hand).**

To find the eigenvalues we can set the characteristic polynomial of A to 0 and solve;

$$\begin{aligned}
\lambda^2 - 2\lambda + (1-\epsilon^2) = 0 \\
\lambda &= \frac{2 \pm \sqrt{4 - 4 + 4\epsilon^2}}{2} \\
\lambda &= 1 \pm \epsilon
\end{aligned}$$

So $\lambda_1 = 1 + \epsilon$, and $\lambda_2 = 1 - \epsilon$.

To find the eigenvector for λ_1, we solve $(A - \lambda_1 I) = 0$;

$$\begin{aligned}
A - (1+\epsilon)I &= \begin{bmatrix} 1-(1+\epsilon) & \epsilon \\ \epsilon & 1-(1+\epsilon) \end{bmatrix} \\
&= \begin{bmatrix} -\epsilon & \epsilon \\ \epsilon & -\epsilon \end{bmatrix} \\
\begin{bmatrix} -\epsilon & \epsilon \\ \epsilon & -\epsilon \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= 0
\end{aligned}$$

We then get the equation $-\epsilon x + \epsilon y = 0$, which shows $x = y$, therefore $v_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

For λ_2 we have $\lambda_2 = 1 - \epsilon$.

$$\begin{aligned}
A - (1-\epsilon)I &= \begin{bmatrix} 1-(1-\epsilon) & \epsilon \\ \epsilon & 1-(1-\epsilon) \end{bmatrix} \\
&= \begin{bmatrix} \epsilon & \epsilon \\ \epsilon & \epsilon \end{bmatrix} \\
\begin{bmatrix} \epsilon & \epsilon \\ \epsilon & \epsilon \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= 0
\end{aligned}$$

We get the equation $\epsilon x + \epsilon y = 0$, which shows $x = -y$, so our eigenvector is $v_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}^T$.

(c) **Now let $\epsilon = \sqrt{\frac{\epsilon_m}{4}}$, where ϵ_m is machine epsilon. Verify that $\epsilon >> \epsilon_m$. Use the Matlab command for finding the coefficients of the characteristic polynomial, *charpoly*, and the Matlab command for finding the roots of a polynomial, *roots*, a the Matlab prompt. The function *charpoly* returns the coefficients of the characteristic polynomial in the correct order for *roots*, i.e., it returns $[a_n, a_{n-1}, \ldots, a_0]$, where a_i is the coefficient for λ^i in the characteristic polynomial.**

Matlab script:

```
epsilon_m = eps;
epsilon = sqrt(epsilon_m / 4);

fprintf('eps = %.16f\n', epsilon);
fprintf('eps m = %.16f\n', epsilon_m);
fprintf('eps >> eps_m: %d\n', epsilon > epsilon_m);

A = [1, epsilon; epsilon, 1];
char_poly = charpoly(A);
disp(char_poly);

eigenvalues = roots(char_poly);
disp(eigenvalues);
```

Our matlab code calculates $char\_poly = \begin{bmatrix}1 & -2 & 1\end{bmatrix}$, and the eigenvalues to be $roots(char\_poly) = \begin{bmatrix}1 & 1\end{bmatrix}^T$. Here, we see the difference of ϵ appear, where we round our eigenvalues down to 1's compared to the values we found in part a, $\begin{bmatrix}1+\epsilon & 1-\epsilon\end{bmatrix}^T$. We see with sufficiently small ϵ Matlab just drops those values off and rounds to 1.

(d) **Do your answers from parts a and b disagree? If so, explain why (use *format long* so that you are able to see by how much they disagree).**

Our answers do agree in parts a and b. In both cases, we are able to simplify our eigenvalues down to $\begin{bmatrix}1 & 1\end{bmatrix}^T$, and the characteristic polynomial coefficients to be $\begin{bmatrix}1 & -2 & 1\end{bmatrix}^T$. Despite this I still tried setting *format long* and rerunning the matlab script to check, but in checking $abs(eigenvalues(1) - eigenvalues(2)) < \epsilon_m$ I see that they are numerically identical there as well.

(e) **Write a Matlab function $[eval, evec] = power\_method(A, x, tol)$ which takes in a matrix A, initial vector for power iteration x, and a tolerance tol, and returns an approximation to the largest eigenvalue $eval$ and the corresponding normalized eigenvector $evec$ using the normalized power method. Use the following criterion for the convergence of power method iteration:**

$$|\lambda^{(k)} - \lambda^{(k-1)}| < tol$$
$$||x^{(k)} - x^{(k-1)}||_2 < tol$$

where $x^{(k)}$ is the approximation to the eigenvector after the kth iteration. where $\lambda^{(k)}$ is the approximation to the eigenvalue after the kth iteration.

```
function [eval, evec] = power_method(A, x, tol)
    diff_lambda = Inf;
    diff_x = Inf;
    x = x / norm(x, 2);
    lambda_prev = 0;

    while diff_lambda > tol || diff_x > tol
        y = A * x;
        x_new = y / norm(y, 2);
        lambda = y' * x;

        diff_x = norm(x_new - x, 2);
        diff_lambda = abs(lambda - lambda_prev);

        x = x_new;
        lambda_prev = lambda;
    end
    eval = lambda;
    evec = x;
end
```

(f) **Keeping $\epsilon = \frac{\sqrt{\epsilon_m}}{4}$ use your function $power\_method$ and the initial guess $x = [3; 4]$ to find the largest eigenvalue of A. Report your approximate eigenvalue. Try doing this with $tol = 10^{-8}, 10^{-9}, 10^{-10}$. Depending on the tolerance, running your script could take a few minutes (not for credit: what does the ratio of the true eigenvalues have to do with it?**

for 10^{-8}, 1.0000000035762784
for 10^{-9}, 1.0000000035885637
for 10^{-10}, 1.0000000037239478

We see that the theoretical largest eigenvalue is increasing as make the tolerance smaller (meaning we want more accuracy).

our ratio is $r = \frac{1-\epsilon}{1+\epsilon}$, which sets the rate at which we converge. We see that we have a ratio where the numerator is decreasing and the denominator is increasing. When r is close to 1 it means we have a small ϵ, so smaller tolerances require more iterations to get a more accurate result.

2. **So far we've only discussed using the power method to find the largest eigenvalue of a matrix. In this problem, we will see how to find the next largest eigenvalue.**

   (a) **Assume a symmetric positive definite matrix A has distinct eigenvalues ($\lambda_i \neq \lambda_j$). Let λ_1 denote the largest eigenvalue and $\overrightarrow{v_1}$ be an associated eigenvector. Show that**

$$B = A - \frac{\lambda_1}{(\overrightarrow{v_1})^T \overrightarrow{v_1}} \overrightarrow{v_1} (\overrightarrow{v_1})^T$$

   has the same eigenvalues and eigenvectors of A, except that λ_1 is replaced by 0, i.e. $B\overrightarrow{v_1} = 0$.

   Hint: It can be shown that the eigenvectors of A are orthogonal, i.e. $(\overrightarrow{v_i})^T \overrightarrow{v_j} = 0$ when $i \neq j$. You can assume this fact, but make sure to explain how you are using it.

   Hint: Be careful with order of vector products. The term $(\overrightarrow{v_1})^T \overrightarrow{v_1}$ is just a dot product and produces a scalar. However, $\overrightarrow{v_1}(\overrightarrow{v_1})^T$ is an outer product that produces a matrix.

   Let $\overrightarrow{v_1}$ be the eigenvector of A with eigenvalue λ_1 s.t. $A\overrightarrow{v_1} = \lambda_1 \overrightarrow{v_1}$. We plug $\overrightarrow{v_1}$ into B:

$$\begin{aligned}
B\overrightarrow{v_1} &= \overrightarrow{v_1}(A - \frac{\lambda_1}{\overrightarrow{v_1}^T \overrightarrow{v_1}} \overrightarrow{v_1}\overrightarrow{v_1}^T) \\
&= A\overrightarrow{v_1} - \frac{\lambda_1}{\overrightarrow{v_1}^T \overrightarrow{v_1}} (\overrightarrow{v_1}^T \overrightarrow{v_1})\overrightarrow{v_1} && A\overrightarrow{v_1} = \lambda_1 \overrightarrow{v_1} \\
&= \lambda_1 \overrightarrow{v_1} - \lambda_1 \overrightarrow{v_1} \\
&= 0
\end{aligned}$$

   Let $\overrightarrow{v_i}$ for any $i \neq 1$ be some other eigenvector of A where the eigenvalue λ_i. We can plug this value into B now:

$$\begin{aligned}
B\overrightarrow{v_i} &= \overrightarrow{v_i}(A - \frac{\lambda_1}{\overrightarrow{v_1}^T \overrightarrow{v_1}} \overrightarrow{v_1}\overrightarrow{v_1}^T) \\
&= A\overrightarrow{v_i} - \frac{\lambda_1}{\overrightarrow{v_1}^T \overrightarrow{v_1}} (\overrightarrow{v_1}^T \overrightarrow{v_i})\overrightarrow{v_1} \\
&= A\overrightarrow{v_i} \\
&= \lambda_i \overrightarrow{v_i}
\end{aligned}$$

   Thus, $B\overrightarrow{v_i} = \lambda_i \overrightarrow{v_i}$ all the other eigenvalues and eigenvectors remain the same when $i \neq 1$.

(b) **Use the previous formula to modify your function** *power_method* **to find the second largest eigenvalue of the** A **from the previous problem,**

$$A = \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix}$$

Note, the largest eigenvalue of B **will be the second largest eigenvalue of** A**.**

```
function [eval, evec] = power_method(A, x, tol)
    diff_lambda = Inf;
    diff_x = Inf;
    x = x / norm(x, 2);
    lambda_prev = 0;

    while diff_lambda > tol || diff_x > tol
        y = A * x;
        x_new = y / norm(y, 2);
        lambda = y' * x;

        diff_x = norm(x_new - x, 2);
        diff_lambda = abs(lambda - lambda_prev);

        x = x_new;
        lambda_prev = lambda;
    end

    lambda1 = lambda;
    v1 = x;
    v1_norm_sq = v1' * v1;
    B = A - (lambda1 / v1_norm_sq) * (v1 * v1');

    diff_lambda = Inf;
    diff_x = Inf;
    x = x / norm(x, 2);
    lambda_prev = 0;

    while diff_lambda > tol || diff_x > tol
        y = B * x;
        x_new = y / norm(y, 2);
        lambda = y' * x;

        diff_x = norm(x_new - x, 2);
        diff_lambda = abs(lambda - lambda_prev);

        x = x_new;
        lambda_prev = lambda;
    end

    eval = lambda;
    evec = x;
end
```

output?-5

3. **Numeric Differentiation**

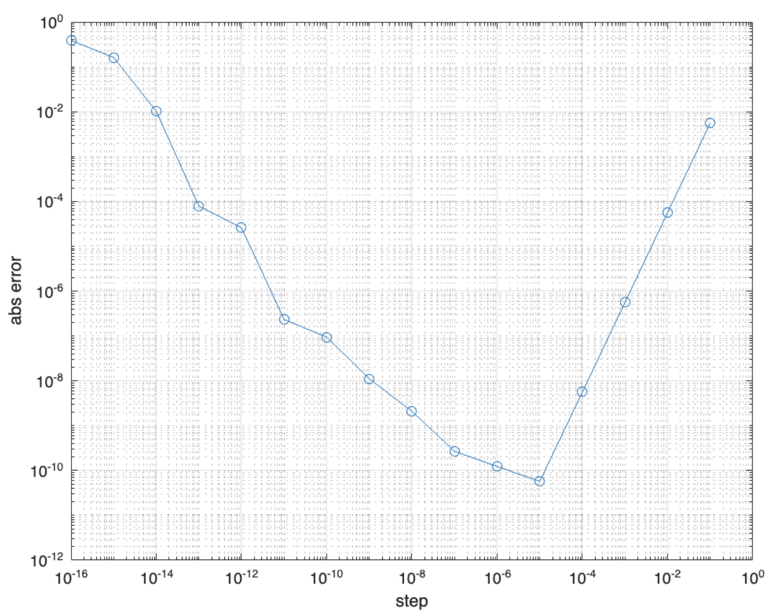
(a) **Use a Taylor series expansion to show that**

$$\frac{f(a+h)-f(a-h)}{2h} = f'(a) + O(h^2)$$

$$\begin{aligned}
f(a+h) &= f(a) + hf'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{6}f^{(3)}(a) + O(h^4) \\
f(a-h) &= f(a) - hf'(a) + \frac{h^2}{2}f''(a) - \frac{h^3}{6}f^{(3)}(a) + O(h^4) \\
f(a+h) - f(a-h) &= 2hf'(a) + \frac{2h^3}{6}f^{(3)}(a) + O(h^4) \\
\frac{f(a+h)-f(a-h)}{2h} &= f'(a) + \frac{h^2}{6}f^{(3)}(a) + O(h^3)
\end{aligned}$$

The leading term is $\frac{h^2}{6}$ so we can simplify using Big-O notation to $O(h^2)$.

(b) **Let $f(x) = \cos(1.5x)$ and $a = 1$. Plot (using the appropriate scaling, i.e. choose the best option between *plot*, *semilogy*, or *loglog*), the absolute error in the approximation of $f'(a)$ by the finite difference in part a, for $h = logspace(-1, -16, 16)$. Explain your results.**



We see the abs error decreases up to the point of a step size of 10^{-5} range, and then we start increasing in error again as we approach 10^{0}. With very small values we get rounding truncation errors that cause a large error, so it is expected for this to improve as we increase the size of our numbers from that starting point.

I chose a *loglog* plot because it shows the relationship between powers changing. In this case, our graph is broken up into a number of straight lines, each of the slopes of those straight lines, x, representing to the power, h^x.

(c) **The error in b arises from two sources of error: truncation error due to the finite taylor series approximation and rounding errors due to floating point operations**

i. **Show that the truncation error is bounded by $C_1 h^2$ for some constant C_1.**

$$\frac{f(a+h)-f(a-h)}{2h} = f'(a) + \frac{h^2}{6} f^{(3)}(a) + O(h^4)$$

We can represent the truncated error as the difference between the finite difference approx and the true value;

$$\frac{h^2}{6} f^{(3)}(a) + O(h^4)$$

We have a bound that $f^{(3)}(a) \leq C$ for some constant, which means we can say

$$\text{Truncation err} \leq \frac{Ch^2}{6} \leq C_1 h^2$$

ii. **Show that the rounding error is bounded by $\frac{C_2 \epsilon}{h}$ where ϵ is the machine precision or machine epsilon and C_2 is some constant.**

Let $\ell(x)$ represent the value we actually compute each time, where $\ell(x) = f(x)(1+g_x)$ where g_x is some value smaller than ϵ. We will then have

$$\frac{\ell(a+h)-\ell(a-h)}{2h} = \frac{f(a+h)-f(a-h)}{2h}(1+g_x) + O(\frac{\epsilon}{h})$$

it is not clear how you go

Again in this case, our rounded error will be bound by the quickest growing term, $\frac{C_2 \epsilon}{h}$. We see the numerator remains relatively constant, while dividing by a small h amplifies it.

iii. **The total error is the sum of the two errors. Add them and write out the expression for the error in your finite difference approximation.**

$$\begin{aligned} \text{trunc error: } &\leq C_1 h^2 \\ \text{rounding err } &\leq \frac{C_2 \epsilon}{h} \\ \text{total err: } &\leq C_1 h^2 + \frac{C_2 \epsilon}{h} \end{aligned}$$

The first term is dominated by the values of h^2, and the second term is dominated for small values of h. The difference here is why the total error gives us a sort of V shape as we transfer from one dominating to the other.

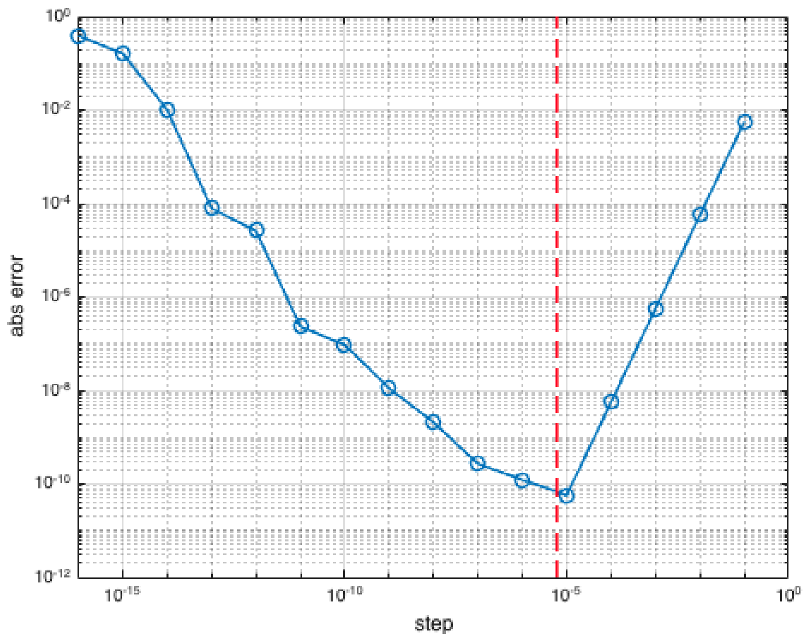
(d) **Use calculus to show that the optimal h to use in the approximation in b, (i.e. the one that gives the smallest error) is $h_{opt} \approx C_{\epsilon^{\frac{1}{3}}}$, where C is some constant. (Hint: Minimize the expression for the total error in c-iii.**

$$
\begin{aligned}
E(h) &= C_1 h^2 + \frac{C_2^{\epsilon}}{h} \\
0 &= 2C_1 h - \frac{C_2^{\epsilon}}{h^2} \\
h^3 &= \frac{C_2^{\epsilon}}{2C_1} \\
h &= (\frac{C_2^{\epsilon}}{2C_1})^{\frac{1}{3}}
\end{aligned}
$$

check your typesetting...

Representing h this way we can show that we want to minimize C. The final $h_{opt} \approx C_{\epsilon^{\frac{1}{3}}}$. The optimal h that will minimize the error depends on the precision we are using and the constant. That is what will ultimately bound our either truncation or rounding errors.

(e) **Does this result agree with the data you plotted in b? Plote a line at $\epsilon^{\frac{1}{3}}$ by using the matlab command** $line([power(eps(\frac{1}{3})power(eps, \frac{1}{3})], [get(gcz,' ylim')])$.



So we see the solution we found above by minimizing our error function does agree with the graph we've created, where we properly identify our minimum value.