Here we present some scaling and accuracy tests for our numerical method. All tests were carried out using Matlab 2022b on an Intel Xeon Gold 6242 CPU. The Matlab file used to perform these tests is included with the example scripts as supplementary material.

The error in K as a function of M:

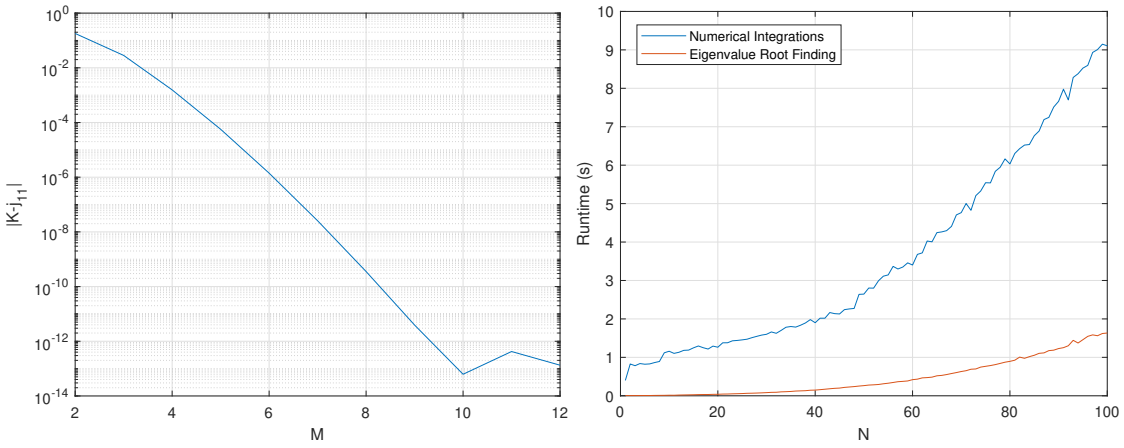
Increasing the number of terms in the expansion, M, results in an increased accuracy for the eigenvalue K. Here, we discuss the dependence of the error in K on the value of M using a case where the analytical solution is known.

We take $(U, a, \beta, R) = (1, 1, 0, \infty)$ in a 1-layer model. Here, the solution can be found analytically using Bessel functions and has a value of $K = j_{11}$ where $j_{11} \approx 3.8317$ is the first non-zero root of the Bessel function J_1. Fig. S1.(a) shows the error in our calculated value of K as a function of the number of terms, M, used in the expansion. We observe that this error decreases exponentially with M before reaching a limit set by the error in the numerical evaluation of the integrals A_{kj} and B_{jk} and the root finding method used to solve the eigenvalue problem. Here, $M = 10$ is sufficient to determine the value of K to 13 significant figures.

Scaling for large N:

While the examples we show are for small values of N, our approach scales well with N and is viable for models with $N \sim O(100)$ layers. Here, we consider an N-layer model with random values of μ and λ. We vary N and determine the average time taken for the N-layer calculation, averaging over 5 runs with different values of μ and λ.

The two limiting step in our method are the numerical integration of A_{kj} and B_{jk} and the root finding step to solve the eigenvalue problem. We evaluate N^2M^2 integrals hence the integration step method is expected to scale as $O(N^2)$. However, determining the integrals to a given level of accuracy taken longest for the diagonal entries, which have the largest values. Therefore, this step often scales close to linearly with N. Additionally, the calculation of A_{kj} and B_{jk} may be easily parallelised in j and k, significantly reducing computation time. The root finding step uses the Matlab 'fsolve' function and is limited primarily by the calculation of a Jacobian matrix. Since the Jacobian can be easily calculated analytically here, we can significantly speed up calculation and increase accuracy by passing the analytical Jacobian to 'fsolve'. This step scales as N^2M^2 however the runtime is dominated by the numerical integration for all cases we consider. Fig. S1.(b) shows the average time taken to calculate a modon solution as a function of N. The times taken for the two limiting steps are shown separately. The average is taken over 5 runs with different parameters. The numerical integration step is parallelised across 32 processes though remains the longest part even though the root finding is carried out on a single process.



(a) Logarithmic plot of the error in calculation of K as a function of the number of terms used, M.

(b) Time taken to calculate the modon solution as a function of the number of layers, N.

Figure S1: Accuracy and scaling test results.