Appendix B Derivation of Brusselator amplitude equations for the hexagonal mode using \texttt{Amp\_solving}

In this section, we will demonstrate a practical application of \texttt{Amp\_solving} to derive the order-3 Brusselator amplitude equations for the hexagonal mode. This amplitude equation allows us to better understand the Turing pattern-forming mechanism, enabling precise control of the patterned structure.

To derive the order-3 amplitude equation, \texttt{Amp\_solving} is modularized into 5 parts:

1. Nonlinear expansion
2. Linear stability analysis
3. Order-2 solvability condition
4. Solving order-2 equation
5. Order-3 solvability condition

The Order-3 solvability condition yields the order-3 amplitude equations. Higher-order amplitude equations can be obtained by iterating the similar procedures of steps 3, 4 and 5: Order-(\(k-1\)) solvability condition \rightarrow Solve order-(\(k-1\)) equation \rightarrow Order-\(k\) solvability condition.

B.1. Nonlinear expansion

B.1.1. Initialize model configuration

Following the work by Pena & Perez-Garcia [2007], we define \(\eta = \sqrt{DX/DY}\). The Brusselator model

\[
\begin{align*}
\frac{\partial X}{\partial t} &= A - (B + 1)X + X^2Y + \eta^2 \nabla^2 X \\
\frac{\partial Y}{\partial t} &= BX - X^2Y + \nabla^2 Y
\end{align*}
\]

has the critical Turing condition \(B_0 = (1 + A\eta)^2\). \(X\) and \(Y\) are functions of temporal and spatial arguments \((t;r)\) in original scales. We will expand the variable to third order in \(\epsilon\):

\[
u(t;r) = u_0 + \epsilon u_1(T_0, T_1, T_2, T_3; R_0, R_1, R_2, R_3) + \epsilon^2 u_2(T_0, T_1, T_2, T_3; R_0, R_1, R_2, R_3) + \epsilon^3 u_3(T_0, T_1, T_2, T_3; R_0, R_1, R_2, R_3) \tag{B.1}
\]

then solve the amplitude equation at order \(\epsilon^3\).

To start the computerized MSE, we first initialize \texttt{Amp\_solving} for the expansion order, system dimension (number of model equations) and multiple-scaled arguments.

```plaintext
> # Initialization
> e_order := 3: # Expansion order
> sys_dim := 2: # System dimensions
> vas := op(op"(seq([R[m],T[m]], m = 0..e_order))); # Variable arguments
> vas := R_0, T_0, R_1, T_1, R_2, T_2, R_3, T_3
```

- \texttt{e_order}: expansion order of the final amplitude equation
- \texttt{sys_dim}: number of model equations
- \texttt{vas}: variable arguments. Temporal and spatial arguments in multiple-scales

B.1.2. Define model equations

In \texttt{Amp\_solving}, model equations are declared via our customized syntax:

- Differential operator conversion
  Temporal derivative \(\partial/\partial t \rightarrow Dt()\); Spatial derivative \(\nabla \rightarrow Ds()\), \(\nabla^2 \rightarrow Ds@2()\).
- Vectorized variables
  Brusselator variables \(X\) and \(Y\) are defined as \texttt{uo1} and \texttt{uo2} respectively. The symbol \(\circ\) means that the
variable has the original temporal and spatial scales. In general, we have the variable definition \( u_{oi} \), in which \( i = 1 \ldots \text{sys\_dim} \). In MAPLE programming, \( u_{oi} \) is equivalent to \( u_{oi||i} \), in which \( || \) is an MAPLE operator to concatenate expressions. The form \( u_{oi||i} \) is useful in a for loop over index \( i \).

- Define constants

  Constants of the model are defined as MAPLE procedures to preserve their independence (see Appendix A.4 for a detailed explanation).

  Following these rules, the Brusselator mode equations are defined as:

\[
\begin{align*}
\# \text{ Define the Brusselator model} \\
\# \# \text{ Activator} \\
PDE[1] & := \text{Dt}(u_{o1}) = A - (B_0 + 1) \cdot u_{o1} + u_{o1}^2 \cdot u_{o2} + N^2 \cdot (D_{s@2})(u_{o1}); \\
A & := () \rightarrow A; \quad \text{# define the constant as a procedure} \\
N & := () \rightarrow N; \\
\# \# \text{ Inhibitor} \\
PDE[2] & := \text{Dt}(u_{o2}) = B_0 \cdot u_{o1} - u_{o1}^2 \cdot u_{o2} + (D_{s@2})(u_{o2});
\end{align*}
\]

- \( B_0 \): the original and unexpanded bifurcation control parameter \( B \)
- \( PDE[i] \): \( i \)-th partial differential equation of the model
- \( N \): \( \eta = \sqrt{D_X/D_Y} \)

**B.1.3. Expansions**

1. Weakly linear perturbation

   Expand the model variable \( u \) as a small perturbation \( \delta u \) around the steady state \( u_0 \):

   \[
   u = u_0 + \delta u
   \]

   Following codes automatically solve \( u_0 \) and substitute the linear expansion to the model equation \( PDE[i] \).

   \[
   \begin{align*}
   \# \text{ Remove temporal and spatial derivatives} \\
   \# \text{ for i from 1 to sys\_dim do} \\
   & \quad \text{PDE\_ss}[i] := 0 = \text{expand(subs(Ds = 0, rhs(PDE[i]))}); \\
   \text{od}; \\
   \# \text{ Automatically solve homogeneous steady-states and store them in ss} \\
   \text{ss} := \text{solve(seq(PDE\_ss[i], i=1..sys\_dim)), seq(u_{oi||i}, i=1..sys\_dim)}; \\
   \text{ss} := \left\{ u_{o1} = A, u_{o2} = \frac{B_0}{A} \right\} \\
   \# \text{ Introduce linear perturbations} \\
   \# \text{ for i from 1 to sys\_dim do} \\
   & \quad u_{oi||i} := u_{oi||i||ij} + \mu_0; \quad \text{# Linear perturbation} \\
   & \quad \text{u}_{oi||i||ij} := \text{rhs(ss[i])}; \quad \text{# Substitute steady-states} \\
   \text{od};
   \end{align*}
   \]

   - \( \delta u \): small perturbation around the steady state \( u_{i\_0} \)

2. Expand differential operators

   (see Appendix A.4 for more details)

   Introduce differential operator expansions Eqs. (7) and (8):

   \[
   \begin{align*}
   \text{Dt} & := \text{sum('e'^m*'D^[2*m+2]', 'm'=0..e\_order)}; \quad \# \text{ Expand temporal derivative} \\
   \text{Ds} & := \text{sum('e'^m*'D^[2*m+1]', 'm'=0..e\_order)}; \quad \# \text{ Expand spatial derivative} \\
   e & := () \rightarrow e; \quad \# \text{ Define scaling constant}
   \end{align*}
   \]

3. Nonlinear expansions

   Introduce the nonlinear variable expansion

   \[
   \delta u = \epsilon u_1 + \epsilon^2 u_2 + \epsilon^3 u_3
   \]
Supplementary notes

and the bifurcation parameter expansion

\[ B = B_0 + \epsilon B_1 + \epsilon^2 B_2 + \epsilon^3 B_3 \]

\[
\text{> for } i \text{ from 1 to sys_dim do} \quad \# \text{Nonlinear variable expansion}
\]
\[
\text{du||i} := \text{sum('e^j*u||i||j', 'j'=1..e_order)};
\]
\[
\text{od:}
\]
\[
\text{> Bo := sum('e^i*B||i', 'i'=0..e_order); \quad \# \text{Bifurcation parameter expansion}
\]
\[
\text{> # Define expanded bifurcation parameters as constants}
\]
\[
\text{> for } i \text{ from 0 to e_order do}
\]
\[
\text{B||i} := \text{subs([variables=(), body=B||i], (variables-> body))};
\]
\[
\text{od:}
\]

- \( u_{ij} \): expanded variables. \( i \) stands for the variable index, \( j \) is the variable expansion index. i.e., for the Brusselator model

\[
\begin{align*}
u_1 &= \begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix}, \\
 u_2 &= \begin{bmatrix} u_{12} \\ u_{22} \end{bmatrix}, \\
 u_3 &= \begin{bmatrix} u_{13} \\ u_{23} \end{bmatrix}
\end{align*}
\]

(4) Obtain order equations

The expanded terms (differential operators, variables and bifurcation parameters) will be substituted back into the model equations automatically. Then we convert the differential expression from D to diff mode.

\[
\text{> # Define alias for compact display}
\]
\[
\text{> for } i \text{ from 1 to sys_dim do}
\]
\[
\text{alias(seq(u||i||j = u||i||j(vas), j=1..e_order));}
\]
\[
\text{od:}
\]
\[
\text{> # Convert D to diff mode for differential equations}
\]
\[
\text{> for } i \text{ from 1 to sys_dim do}
\]
\[
PDE\text{sorted}[i] := \text{simplify(collect(PDE[i],e), e^(e_order+1)=0)};
\]
\[
PDE\text{diff}[i] := \text{convert(PDE\text{sorted}[i](vas), diff)};
\]
\[
\text{od:}
\]

- \( PDE\text{sorted}[i] \): fully expanded model equation up to the order \( e\cdot\text{order} \)
- \( PDE\text{diff}[i] \): \( PDE\text{sorted}[i] \) in the diff form

Using \text{coeff()} to extract coefficients at a specific order of \( \epsilon \) from both sides of \( PDE\text{diff}[i] \), we can obtain a series of raw order equations. This is done automatically in \text{Amp_solving} by scanning each model equation at all \( \epsilon \) orders.

\[
\text{> # Collect terms with the same order of e}
\]
\[
\text{> for } i \text{ from 1 to sys_dim do}
\]
\[
\text{for } j \text{ from 1 to e_order do} \# \text{ith equation at order } j
\]
\[
eq|\text{eq||i||order||j} :=
\]
\[
\text{expand(coeff(lhs(PDE\text{diff}[i]), e,j)) = coeff(rhs(PDE\text{diff}[i]), e,j));}
\]
\[
\text{od:}
\]

- \( eqi\_orderj \): raw order equation, read as the \( i \)-th equation of \( \epsilon^j \). It is obtained by equating the coefficients of \( \epsilon^j \) from both sides of the expanded equation.

We need to rearrange the raw order equation \( eqi\_orderj \) to the form as Eqs. (10), (13) and (21), so that the LHS of the order equation has only the linear operation \((\partial/\partial T_0 - L_c)\) on the current order variables. We use \text{selectremove()} to select the current order variables then group them at the LHS while placing other terms at the RHS (see Sec. 4.1 for more details). The sorted order equation is read \( eqi\_orderj\_diff \), the suffix of which \text{diff} indicates its explicit differentiation display.

\[
\text{> # Tidy up order equations}
\]
\[
\text{> for } i \text{ from 1 to sys_dim do}
\]
\[
\text{for } j \text{ from 1 to e_order do}
\]
\[
PDE\text{temp||i||j} := \text{lhs(eq||i||order||j)} - \text{rhs(eq||i||order||j)};
\]
At last, we vectorize the RHS of order equations at each $\epsilon$ order to a column vector (e.g., $I_j$ in Eqs. (13) and (21)). This is required for the scalar product of the solvability condition. At the expansion order $j$, RHS for all order-$j$ equations are stored in $\text{eq}_\text{vec}_\text{order}_j$ (representing $I_j$).

\begin{verbatim}
> # Vectorize rhs of order equations
> for j from 1 to n_order do
> eq_vec_order[j] := Vector[column]([seq(rhs(eq||i||order||j||diff), i = 1..sys_dim)]);
> end:
\end{verbatim}

B.2. Linear stability analysis

The order-1 equation recovers the linear stability analysis (LSA), which is coded in a separate MAPLE worksheet LSA.mw. To do this, $\text{Amp solving}$ will save the order-1 equation to a .txt file:

\begin{verbatim}
> eqlist_order1 := seq(eq||i||order1, i = 1..sys_dim);
> save eqlist_order1, n_order, sys_dim, vas, 'eqlist_order1.txt' :
\end{verbatim}

LSA in LSA.mw is coded as following:

\begin{verbatim}
> # LSA.mw
>
> read "eqlist_order1.txt";
> # convert from diff to D form prepared for Jac calculation
> eqlist := convert(eqlist_order1, D):
> # Remove function properties required for deriving the Jac matrix
> eqlist_subs := subs(seq(D[2](u||i||1)(vas) = D[2](u||i||1), i=1..sys_dim),
    seq(D[1,1](u||i||1)(vas) = D[1,1](u||i||1), i=1..sys_dim),
    seq((u||i||1)(vas) = (u||i||1), i=1..sys_dim), eqlist);
> # Jacobian matrix
> Jac_list := [seq(rhs(eqlist_subs[i]), i=1..sys_dim)]: # eq list
> Jac-vars := [seq(u||i||1, i=1..sys_dim )]:
> Jac := VectorCalculus[Jacobian](Jac_list, Jac-vars):
> # Pure Turing condition: vanished eigenvalue
> eig_real := remove(hastype, expand(eig_Jac[1]), sqrt); # real part of the dom eig
> eig_sqrt := select(hastype, expand(eig_Jac[1]), sqrt); # imag part of the dom eig
> eig_zero := eig_real = eig_sqrt: # force to get zero eigenvalue
> B_T := expand(solve(eig_zero, B_0)); # B is a function of q
>
> # At the critical wavenumber
> q_eq := convert(select(has, B_T, q), `list`):
> q_eq[1] := solve(q_eq[1] = q_eq[2], q): # only choose the positive solution
> # Critical wavenumber
> Bc_T := simplify(subs(B_0 = Bc_T = 0, q = qc_T, Jac_ss));
> RT, eig_T := MTM[eig](Jac_T):
> R_T := RT(1..2, 2)/(RT(1,2)): # Turing right eigenvector
\end{verbatim}
> Jac_T_Tr := MTM[transpose](Jac_T);
> LT, L2 := MTM[eig](Jac_T_Tr);
> L_T := MTM[transpose](LT(1..2, 1))/(LT(1,1)); # Turing left eigenvector
>
> save R_T, L_T, Bc_T, qc_T, `Brusselator_Tsets.txt`;

From LSA.txt, the critical Turing condition, the critical wavenumber, and critical right and left eigenvectors are calculated and saved in a standard .txt file that can be loaded in the main program by using the read command.

> read "Brusselator_Tsets.txt";  # read LSA results
> Tsets := {
  seq(R_T||i = R_T(i), i=1..sys_dim), # right eigenvectors
  seq(L_T||i = L_T(i), i=1..sys_dim), # left eigenvectors
  B_0 = Bc_T # Turing condition
};
> Tsets_extra := q0 = qc_T; # critical wavenumber

Note that we define the critical eigenvectors and wavenumber implicitly in Maple sets since we only need their explicit expressions when necessary. e.g., we want to extract the coefficient of \(e^y\) from the expression \(f\):

> f := (y*exp(-y) + x*exp(2*y))^2;
> exp_coeff := coeff(expand(f), exp(y));

Suppose that \(x\) and \(y\) are functions of \(a\) and \(b\), we can define a comma-separated set to include their expressions:

> fun_set := y = 2*a + 5*b, x = (a + b)^2:

To obtain the explicit form of \(exp\_coeff\), we simply substitute \(fun\_set\) into it:

> subs(fun_set, exp_coeff)

If \(x\) and \(y\) are declared explicitly before \(f\), we are not able to extract the coefficient of \(e^y\) since it will automatically become \(e^{2a+5b}\) in \(f\). If \(x\) and \(y\) are declared explicitly after \(f\), we can extract the coefficient of \(e^y\), but \(f\) is not in its original form anymore. This idea is important to define the structure ansatz since it is a function of the wavenumber \(q_0\), more specifically, it is an exponential function of \(q_0\): e.g., \(e^{i\tilde{q}_1r}\), \(|\tilde{q}_1| = q_0\). In the solvability condition, we need to extract the coefficient of exponential functions, thus we must keep \(q_0\) as a symbol. Moreover, this technique enhances computing efficiency dramatically since we can substitute the explicit expressions when necessary.

B.3. Order-2 solvability condition

B.3.1. Structure ansatz

The structure ansatz for the hexagonal mode is written as:

\[
\mathbf{u}_1 = A_{T_1} e^{i\tilde{q}_1r} \mathbf{R}_T + A_{T_2} e^{i\tilde{q}_2r} \mathbf{R}_T + A_{T_3} e^{i\tilde{q}_3r} \mathbf{R}_T + \text{c.c.}
\]

where c.c. stands for complex conjugate pairs. \(\tilde{q}_1\), \(\tilde{q}_2\), and \(\tilde{q}_3\) are three critical wavevectors with identical modulus (equal to the critical wavenumber \(q_0\)) and 120° angular separation. One wavevector coincidences with the resultant wavevector of other two. In \texttt{amp-solving}, we state this condition as \texttt{q_loop}:

\[
\tilde{q}_1 + \tilde{q}_2 = -\tilde{q}_3, \quad \tilde{q}_1 + \tilde{q}_3 = -\tilde{q}_2, \quad \tilde{q}_2 + \tilde{q}_3 = -\tilde{q}_1.
\]

(B.2)

In the meantime, the product of any two wavevectors has the \texttt{q_times} relationship:

\[
\tilde{q}_1 \cdot \tilde{q}_2 = \tilde{q}_1 \cdot \tilde{q}_3 = \tilde{q}_2 \cdot \tilde{q}_3 = q_0^2 \cos(\theta), \quad \theta = 120^\circ
\]

(B.3)
At1, At2 and At3 are mode amplitudes. The mode amplitude describes the mode dynamics in slow temporal and extended spatial scales, thus depending on scaled arguments $T_1, T_2; R_1, R_2$, that is, aas (amplitude arguments) in the program.

\begin{verbatim}
> aas := op(op~([seq([R[j],T[j]], j = 1..e_order)])); # Amplitude arguments
> # Right and left eigenvectors
> RT := Vector[column]([seq(R||i, i=1..sys_dim)]):
> LT := Vector[column]([seq(LT||i, i=1..sys_dim)]):
> # Apply alias for compact amplitude display
> alias(AT1=A_T1(aas), AT1c=A_T1c(aas)):
> alias(AT2=A_T2(aas), AT2c=A_T2c(aas)):
> alias(AT3=A_T3(aas), AT3c=A_T3c(aas)):
> # Wavevector conditions for the hexagonal mode
> qsubs := subs(Tsets_extra, [q1=q0, q2=q0, q3=q0]):
> q_times := [q1*q2 = q0^2*(-1/2), q1*q3 = q0^2*(-1/2), q2*q3 = q0^2*(-1/2)]:
> q_loop := q2*q3=-q1, q1*q3=-q2, q1*q2=-q3:
> # Order 1 ansatz
> ans1 := AT1*exp(I*q1*R[0])*RT + AT2*exp(I*q2*R[0])*RT + AT3*exp(I*q3*R[0])*RT + AT1c*exp(-I*q1*R[0])*RT + AT2c*exp(-I*q2*R[0])*RT + AT3c*exp(-I*q3*R[0])*RT;

• At1c, At2c, At3c: $A_{\tau_1}, A_{\tau_2}, A_{\tau_3}$, complex conjugate pairs.

B.3.2. Order-2 solvability condition
Recall the column vector eq_vec orderj (where j represents the expansion order, i.e., $I_j$ in Eqs. (13) and (21)). Here, we apply the DotProduct (from VectorCalculus package) between the critical left eigenvector LT and the RHS of the order-2 equations eq_vec_order2 for the orthogonal condition (see. Eq. (16)). According to Eqs. (37), the solvability condition yields a series of constraints that requires us to collect secular terms and force them to zero.

\begin{verbatim}
> # Orthogonal condition
> SC2 := VectorCalculus[DotProduct](LT, eq_vec_order2):
> # Substitute order 1 ansatz
> SC2_temp1 := PDETools[daubs]([seq(u||i||1(vas)=ans1[i], i=1..sys_dim)], SC2):
> SC2_temp2 := simplify(expand(SC2_temp1)):
> # Substitute wavevector conditions for the hexagonal mode
> SC2_temp3 := subs(q_loop, SC2_temp2):
> # Collect secular terms and force them to zero
> for i from 1 to 3 do
>   SC2_temp4||i := coeff(simplify(SC2_temp3), exp(I*q||i*R[0]));
>   SC2_temp4c||i := coeff(simplify(SC2_temp3), exp(-I*q||i*R[0]));
>   SC2_temp5||i := collect(simplify(SC2_temp4||i), B_1, diff, 'distributed');
>   SC2_temp5c||i := collect(simplify(SC2_temp4c||i), B_1, diff, 'distributed');
>   SC2_temp6||i := collect(simplify(subs(Tsets, SC2_temp5||i)), B_1, diff, AT);  
>   SC2_temp6c||i := collect(simplify(subs(Tsets, SC2_temp5c||i)), B_1, diff, AT);
> # Eliminate secular terms
> SC2_Case||i := denom(simplify(SC2_temp6||i))*simplify(SC2_temp6||i) = 0;
> SC2_Casecc||i := denom(simplify(SC2_temp6c||i))*simplify(SC2_temp6c||i) = 0;
> # Solve B_1 from above equations
> B1_subscase||i := B_1 = solve(SC2_Case||i, B_1);
\end{verbatim}
The for loop in the above code extracts all six solvability conditions (i.e., Eqs. (37)), yielding six constraints: SC2_Case1, SC2_Case2 and SC2_Case3; conjugate pairs SC2_Casecc_1, SC2_Casecc_2 and SC2_Casecc_3. By examining the constraint SC2_Case1:

\[
\begin{align*}
    & \frac{\partial}{\partial T_1} A B_1 - \left( \frac{\partial}{\partial T_1} A T_1 \right) A - \left( \frac{\partial}{\partial T_1} A T_1 \right) A^2 N + \left( \frac{\partial}{\partial T_1} A T_1 \right) A N^2 \\
    & + \left( \frac{\partial}{\partial T_1} A T_1 \right) A^2 N^3 - 2 AT_2c AT_3c N^2 A^2 + 2 AT_2c AT_3c = 0
\end{align*}
\]

we see that the mode amplitude \( A_{T1} \) does not depend on \( T_1 \) or \( R_1 \), thus we can re-define the amplitude arguments aas:

\[ aas := R[2], T[2], R[3], T[3]; \]

Now SC2_Case_1 outputs:

\[ SC2_{\text{Case}1}; \]

\[ AT_1 A B_1 - 2 AT_2c AT_3c N^2 A^2 + 2 AT_2c AT_3c = 0 \]

B_1 is solved from above equation and stored in B1_subsCase1.

### B.4. Solution of order-2 equation

Amp_solving follows the exponential pattern matching method (see Secs. 3.4 and 4.2 for detailed explanations) to solve the order equation, which we will demonstrate in the following five sections.

#### B.4.1. Expand the RHS of the order-2 equation

To construct coefficient equations, we need to first determine the precise structure of the order-2 solution, which can be derived by examining the fully expanded order-2 equation. To do this, we will substitute the order-2 solution structure and the order-2 solvability conditions into the order-2 equation.

\[ \text{for } j \text{ from 1 to sys_dim do} \]

\[ \text{# Subs structure ansatz} \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz1} := \text{PDETools[dsubs]}( \text{seq(u}||i,\text{vas})=\text{ans}1[i], i=1\ldots\text{sys_dim}), \]

\[ \text{eq}||j,\text{order2}_\text{diff1}: \]

\[ \text{# Subs Turing conditions} \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz2} := \text{subs(op(Tsets), eq}||j,\text{order2}_\text{withAnsatz1}); \]

\[ \text{# Subs q_loop condition} \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz3} := \text{subs(q_loop, simplify(expand(eq}||j,\text{order2}_\text{withAnsatz2})) }); \]

\[ \text{# Collect exp functions} \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz4} := \text{collect(eq}||j,\text{order2}_\text{withAnsatz3, exp}); \]

\[ \text{# Subs order-2 solvability conditions} \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz} := \text{subs(seq(-SC2_Case}||i1, i=1\ldots3), seq(-SC2_Casecc}||i1, i=1\ldots3), \]

\[ \text{eq}||j,\text{order2}_\text{withAnsatz4}); \]

\[ \text{od}; \]

#### B.4.2. Determine the structure of the order-2 solution

Amp_solving uses op() to extract unique exponential components from the fully expanded order-2 equation eqj_order2_withAnsatz, then produces a series of coefficients ci_j to determine the structure of the order-
2 solution.

> # Extract exp components
> exp_fun := op(select(has, indets(simplify(expand(eq||1||order2_withAnsatz))), exp));
> 
> # Unknown coefficient cij, i-ith equations, j-jth coefficient
> cas := aas;
> for i from 1 to sys_dim do
>     seq( alias(c||i||j=c||i||j(cas)), j=1..nops([exp_fun])+1 );
> od:
> 
> # Order 2 ansatz
> for i from 1 to sys_dim do
>     ans2[i] := sum('c||i||j * exp_fun[j]', 'j'=1..nops([exp_fun]) +
>     c||i||(nops([exp_fun])+1):
> od:
>
It should be mentioned that coefficient c_{i,j} is not a constant but a function having the same dependence as the mode amplitude, so we name the coefficient dependence cas (coefficient arguments). It should also be noticed that the order-2 solution structure ans2 and structure ansatz ans1 share the same dependence vas.

B.4.3. Substitute order-2 solution structure into the LHS of the order-2 equation

> for i from 1 to sys_dim do
>     # Subs order 2 ansatz into the LHS
>     eq||i||lhs_temp1 := collect( PDETools[dsubs]([seq(u||i||2(vas)=ans2[i], i=1..sys_dim)],
>         lhs(eq||i||order2_withAnsatz), exp):
>     
>     # Subs q.times conditions into the LHS
>     eq||i||lhs_temp2 := collect(simplify(simplify(eq||i||lhs_temp1, q_times)), exp):
>     
>     # Re-construct the order 2 equation
>     eq||i||order2_subs := eq||i||lhs_temp2 = rhs(eq||i||order2_withAnsatz):
> od:

B.4.4. Construct coefficient equations by matching exponential patterns

Now, the order-2 equation has been fully expanded with both sides in the summation of exponential series (can be checked by calling eq1_order2_subs or eq2_order2_subs).

In Sec. 3.4, we see that the exponential series of the expanded order-2 equation starts from the zero power. Amp_solving treats separately these exponential patterns that have nonzero and zero exponential powers since op() cannot extract terms with zero exponential power (i.e., $e^0$) directly. Assuming the model has i equations and j nonzero exponential components, there will be a total of $2(i \cdot j + i)$ coefficient equations.

> for i from 1 to sys_dim do
>     # Coefficient equations for nonzero exp power
>     for j from 1 to nops([exp_fun]) do
>         eq||i||l_coef||j :=
>             coeff( lhs(eq||i||order2_subs), exp_fun[j] ) =
>             coeff( rhs(eq||i||order2_subs), exp_fun[j] );
>     od:
> 
>     # Coefficient equations for zero exp power
>     eq||i||l_coef||(nops([exp_fun])+1) :=
>         remove(has, lhs(eq||i||order2_subs), seq(exp_fun[j], j=1..nops([exp_fun])) ) =
>         remove(has, rhs(eq||i||order2_subs),
Supplementary notes

B.4.5. Solve coefficient equations

To solve unknown coefficients $c_{i,j}$ in the order-2 solution structure $\text{ans2}$, $\text{Amp} \_ \text{solving}$ will group coefficient equations in $\text{eq} \_ \text{list}$ and unknown coefficients in $\text{coef} \_ \text{list}$, then feed them to the MAPLE solver $\text{solve}()$ to obtain the solution $\text{coeff} \_ \text{order2}$.

```maple
> # List of coefficient equations
> eq_list := [seq(seq(eq||i||j, i=1..sys_dim), j=1..nops([exp_fun])+1 )]:
> # List of unknown coefficients
> coef_list := [seq(seq(c||i||j, i=1..sys_dim ), j=1..nops([exp_fun])+1 )]:
> # Solve coefficients
> coeff_order2_temp := solve(eq_list, coef_list):
> coeff_order2 := subs(Tsets_extra, subs(qsubs, coeff_order2_temp ) );
```

B.5. Order-3 solvability condition

The order-3 solvability condition is derived in the same fashion as the order-2 solvability condition: Substitute the structure ansatz and order-2 solution into the RHS of the order-3 equation $\text{eq} \_ \text{vec} \_ \text{order3}$, then apply the orthogonal condition to find secular terms and force them to zero. The order-3 amplitude equation appears here.

B.5.1. Substitute the structure ansatz and order-2 solution into the order-3 equation

The explicit order-2 solution $u_2$ is obtained by plugging solved coefficients $\text{coeff} \_ \text{order2}$ into the corresponding ansatz $\text{ans2}$.

```maple
> # Explicit order-2 solution
> ans2_subs := subs(op(op(coeff_order2)), seq(ans2[i], i=1..sys_dim)):
> # Subs the structure ansatz and order-2 solution into the RHS of the order-3 equation
> eq_vec_order3_subs := subs(
    [seq(u||i||1(vas)=ans1[i], i=1..sys_dim)],
    [seq(u||i||2(vas)=ans2_subs[i], i=1..sys_dim)], eq_vec_order3):
```

B.5.2. Apply the orthogonal condition

```maple
> # Orthogonal condition
> SC3 := VectorCalculus[DotProduct](LT, eq_vec_order3_subs):
> # Collect exp components
> SC3_temp1 := collect(simplify(SC3), exp):
> # Subs q_loop condition
> SC3_temp2 := subs(q_loop, SC3_temp1):
> # Extract secular terms
> SC3_temp3 := simplify( subs(Tsets, coeff(SC3_temp2, exp(I*q1*R[0]))) ) = 0:
> # Tidy up the amplitude equation for the q1 mode
> SC3_temp4 := collect(SC3_temp3, diff,factor):
```

The hexagonal mode has 6 amplitude equations relating to $A_{T1}$, $A_{T2}$ and $A_{T3}$; $A_{T1}^+$, $A_{T2}^+$ and $A_{T3}^+$. Here, we derive the amplitude equation only with respect to $A_{T1}$ (i.e., $\text{SC3} \_ \text{temp4}$). Amplitude equations about $A_{T2}$ and $A_{T3}$ can be obtained by permutations of indices. We then tidy up $\text{SC3} \_ \text{temp4}$ and return its argument scales.
B.5.3. Tidy up amplitude equation

The following algorithm will automatically sort out the raw amplitude equation into the form of Eq. (24): Find the derivative term and place it at the LHS, keeping other terms at the RHS.

> # Extract the coefficient of the temporal derivative
> AT_coeff := PDEtools[dcoeffs](lhs(SC3_temp4), AT1)[1]:
> # Normalize the coefficient of the temporal derivative Eq.
> SC3_temp5 := collect(expand(SC3_temp4/(AT_coeff)), diff,factor):
> # Select the temporal derivative
> time_diffT := selectfun(SC3_temp5, diff)[1]:
> # Place the temporal derivative at LHS, other terms at RHS
> SC3_temp6 := time_diffT = -selectremove(has, lhs(SC3_temp5), time_diffT)[2]:
> # AT1_abs: modulus of the amplitude AT1, i.e., |AT1|.
> SC3_temp7 := simplify(SC3_temp6, AT1*AT1c = AT1
> AT2*AT2c = AT2_abs^2, AT3*AT3c = AT3_abs^2):
> # Tidy up the equation
> SC3_temp8 := collect(SC3_temp7, AT1_abs, AT2_abs, AT3_abs, factor):

$$\frac{\partial}{\partial T_2} \text{AT1} = \frac{1}{9} \left[ \begin{array}{c} A T_1 (A N + 2) (8 N^2 A^2 + 4 - 21 A N) A T_1 \epsilon^2 \\ - A T_1 (3 A^3 N^3 - 7 N^2 A^2 - 5 A N + 3) A T_2 \epsilon^2 \\ - A T_1 (3 A^3 N^3 - 7 N^2 A^2 - 5 A N + 3) A T_3 \epsilon^2 \\ - 2 B_1 A T_1 \epsilon A T_2 c + B_2 A A T_1 \epsilon \\ \end{array} \right]$$

B.5.4. Return arguments to normal scales

Assume $Z_1 = \epsilon A T_1$ (representing amplitude scaling), where $Z_1$ is the amplitude carrying unscaled arguments ($r_0, T_0$), namely $nas$: normal arguments. Meanwhile considering $T_2 = \epsilon^2 T_0$ (temporal argument scaling), the LHS of $SC3_temp8$ turns to be $1/\epsilon^3 \cdot \partial Z_1/\partial T_0$. Similarly, we have the following conversions:

$$\begin{align*}
A T_1 &= Z_1/\epsilon & A T_2 &= Z_2/\epsilon & A T_3 &= Z_3/\epsilon \\
A T_1^\epsilon &= Z_1^\epsilon/\epsilon & A T_2^\epsilon &= Z_2^\epsilon/\epsilon & A T_3^\epsilon &= Z_3^\epsilon/\epsilon
\end{align*}$$

(B.4)

Substituting these relationships into $SC3_temp8$ generates the $Z_1$ amplitude equation with normal scales.

> nas := R[0], T[0]; # normal arguments
> alias(Z1=Z_1(nas), Z1c=Z_1c(nas));
> alias(Z2=Z_2(nas), Z2c=Z_2c(nas));
> alias(Z3=Z_3(nas), Z3c=Z_3c(nas));
> SC3_temp9 := 1/\epsilon^3 * diff(Z1, T[0]) = subs(
  AT1 = Z1/\epsilon, AT2 = Z2/\epsilon, AT3 = Z3/\epsilon,
  AT1c = Z1c/\epsilon, AT2c = Z2c/\epsilon, AT3c = Z3c/\epsilon,
  AT1_abs^2 = Z1_abs^2/\epsilon^2, AT2_abs^2 = Z2_abs^2/\epsilon^2,
  AT3_abs^2 = Z3_abs^2/\epsilon^2, rhs(SC3_temp8)):
> SC3_temp10 := simplify(e^3*SC3_temp9):
> SC3_temp11 := collect(SC3_temp10, Z1_abs, Z2_abs, Z3_abs, factor):

$$\frac{\partial}{\partial T_0} Z_1 = -\frac{1}{9} \left[ \begin{array}{c} Z_1 (A N + 2) (8 N^2 A^2 + 4 - 21 A N) Z_1 \epsilon^2 \\ - Z_1 (3 A^3 N^3 - 7 N^2 A^2 - 5 A N + 3) Z_2 \epsilon^2 \\ \end{array} \right]$$
The critical right and left eigenvectors associated with the Turing eigenvalue \( q = 0 \) twice, respectively at a zero wavenumber lead the system to a codimension-2 point (CTHP), where the real part of the dominant eigenvalue touches of the two bifurcation parameters. The Turing–Hopf (TH) bifurcation for the Brusselator model can be triggered by the simultaneous tuning mode using Appendix C Derivation of Brusselator amplitude equations for the Turing–Hopf. These literatures also provide further analysis of the amplitude equations. This result is consistent with the work by Pena & Perez-Garcia [2007, 2001], and Verdasca et al. [1992].

**Appendix C Derivation of Brusselator amplitude equations for the Turing–Hopf mode using \texttt{Amp.solving}**

The Turing–Hopf (TH) bifurcation for the Brusselator model can be triggered by the simultaneous tuning of the two bifurcation parameters \( B \) and \( \sigma = D_X/D_Y \). Critical settings

\[
B = B_0 + 1 + A^2 \\
\sigma = \sigma_0 = \frac{(-1 + \sqrt{A^2 + 1})^2}{A^2}
\]

lead the system to a codimension-2 point (CTHP), where the real part of the dominant eigenvalue touches zero twice, respectively at a zero wavenumber \( q = 0 \) at eigenvalue \( \lambda_c^H = 0 + i\omega_c \) and at a nonzero wavenumber \( q_e = \sqrt{\frac{A}{\sigma_0}}^{1/4} \) with eigenvalue \( \lambda_c^T = 0 + i0 \).

The critical right and left eigenvectors associated with the Hopf eigenvalue \( \lambda_c^H \) are

\[
\mathbf{R}_H = \begin{bmatrix} \frac{1}{i - A} \\ A \end{bmatrix}, \quad \mathbf{L}_H = \begin{bmatrix} 1 \frac{A(A + i)}{1 + A^2} \end{bmatrix}, \quad i = \sqrt{-1}
\]

The critical right and left eigenvectors associated with the Turing eigenvalue \( \lambda_c^T \) are

\[
\mathbf{R}_T = \begin{bmatrix} \frac{1}{-\sqrt{\sigma_0}} \\ \frac{A}{A(1 + A^{1/4})} \end{bmatrix}, \quad \mathbf{L}_T = \begin{bmatrix} 1 \frac{A\sqrt{\sigma_0}}{(1 + A\sqrt{\sigma_0})} \end{bmatrix}
\]

For a codimension-2 bifurcation, we need to expand both bifurcation parameters simultaneously:

\[
B = B_0 + \epsilon B_1 + \epsilon^2 B_2 + \ldots \\
\sigma = \sigma_0 + \epsilon \sigma_1 + \epsilon^2 \sigma_2 + \ldots
\]
Now, we will utilize Amp_solving to derive the order-3 amplitude equation at a CTHP. We first initialize the algorithm:

```latex
> # Initialization
> e_order := 3: # Expansion order
> sys_dim := 2: # System dimensions
> vas := op(op'([seq([R[m],T[m]], m = 0..e_order)])); # Variable arguments
  vas := R_0, T_0, R_1, T_1, R_2, T_2, R_3, T_3
```

then define the Brusselator model ($\sigma_0$ is represented by $\text{No}$ in the code)

```latex
> # Define the Brusselator model
> ## Activator
> PDE[1] := Dt(uo1) = A - (Bo + 1) * uo1 + uo1^2 * uo2 + No * (Ds@@2)(uo1);
> A := ()->A: # define the constant as a procedure
> ## Inhibitor
> PDE[2] := Dt(uo2) = Bo * uo1 - uo1^2 * uo2 + (Ds@@2)(uo2);
```

Amp_solving starts the expansion by introducing a linear perturbation:

```latex
> # Remove temporal and spatial derivatives
> for i from 1 to sys_dim do
>   PDE_ss[i] := 0 = expand(subs(Ds = 0, rhs(PDE[i])));
> od:
> # Automatically solve homogeneous steady-states and store them in ss
> ss := solve( seq(PDE_ss[i], i=1..sys_dim),
>              seq(uo||i, i=1..sys_dim) );
> # Introduce linear perturbations
> for i from 1 to sys_dim do
>   uo||i := u||i||0 + du||i; # Linear perturbation
>   u||i||0 := rhs(ss[i]); # Substitute steady-states
> od:
```

followed by expanding differential operators

```latex
> Dt := sum('e^m*D[2*m+2]', 'm'=0..e_order): # Expand temporal derivative
> Ds := sum('e^m*D[2*m+1]', 'm'=0..e_order): # Expand spatial derivative
> e := ()->e: # Define scaling constant
```

and model variables perturbations and bifurcation parameters,

```latex
> for i from 1 to sys_dim do
>   du||i := sum('e^j*u||i||j', 'j'=1..e_order);
> od:
> Bo := sum('e^i*B||i', 'i'=0..e_order); # Bifurcation parameter expansion
> No := sum('e^i*N||i', 'i'=0..e_order);
> # Define expanded bifurcation parameters as constants
> for i from 0 to e_order do
>   B||i := subs([variables=(), body=B||i], (variables-> body));
>   N||i := subs([variables=(), body=N||i], (variables-> body));
> od:
```

The order equations can be obtained by the following codes:

```latex
> for i from 1 to sys_dim do
>   alias(seq(u||i||j||j = u||i||j(vas), j=1..3));
> od:
```
The critical bifurcation conditions are calculated in an external worksheet LSA_TH.mw then loaded to the main program (critical frequency $\omega_c$ is represented by $w$ in the code):

```maple
> read "Brusselator_THsets.txt": # read LSA results, created by LSA_TH.mw
> Tsets := {
    seq(RH||i = R_H(i), i=1..sys_dim), # Hopf right eigenvectors
    seq(RH||i||c = R_Hc(i), i=1..sys_dim), # Hopf right eigenvectors (c.c.)
    seq(RT||i = R_T(i), i=1..sys_dim), # Turing right eigenvectors
    seq(LH||i = L_H(i), i=1..sys_dim), # Hopf left eigenvectors
    seq(LH||i||c = L_Hc(i), i=1..sys_dim), # Hopf left eigenvectors (c.c.)
    seq(LT||i = L_T(i), i=1..sys_dim), # Turing left eigenvectors
    B_0 = Bc_TH, # critical bifurcation setting
    q0 = qc_T, # critical wavenumber
    THsets_extra := w = A, N_0 = sc_TH;
};
```

The structure ansatz is defined according to Eq. (33):

```maple
> aas := op(op"[(seq([R[j],T[j]], j=1..e_order))]);
> RT := Vector[column]([seq(RT||i, i=1..sys_dim)]): # Turing right eigenvector
> LT := Vector[column]([seq(LT||i, i=1..sys_dim)]): # Turing left eigenvector
> RH := Vector[column]([seq(RH||i, i=1..sys_dim)]): # Hopf right eigenvector
> RHC := Vector[column]([seq(RH||i||c, i=1..sys_dim)]): # Hopf right eigenvector (c.c.)
> LH := Vector[column]([seq(LH||i, i=1..sys_dim)]): # Hopf left eigenvector
> LHC := Vector[column]([seq(LH||i||c, i=1..sys_dim)]): # Hopf left eigenvector (c.c.)
> alias(AT=A_T(aas), ATc=A_Tc(aas)): # Turing amplitude
> alias(AH=A_H(aas), AHc=A_Hc(aas)): # Hopf amplitude
> anal := AT*exp(I*q0*R[0])*RT + ATc*exp(-I*q0*R[0])*RT +
        AH*exp(I*u*T[0])*RH + AHc*exp(-I*u*T[0])*RHC;
```
Following Eqs. (39a) and (39b), the TH mode solvability condition has separated Turing and Hopf components:

> # Solvability condition - Turing
> SC2T := VectorCalculus[DotProduct](LT, eq_{vec_order2}): 
> SC2T_temp1 := PDETools[dsubs](\[seq(u_{i1}(vas)=ans1[i], i=1..sys_dim)\], SC2T): 
> SC2T_temp2 := simplify(expand(SC2T_temp1)): 
> SC2T_temp3 := coeff(SC2T_temp2, exp(I*q0*R[0]0)): 
> SC2T_temp4 := collect(simplify(SC2T_temp3), B[1], diff, 'distributed'): 
> SC2T_temp5 := collect(simplify(subs(subs(THsets, SC2T_temp4))), B[1], diff, AT): 
> SC2T_temp6 := denom(simplify(SC2T_temp5)) * simplify(SC2T_temp5) = 0: 
> SC2T temp6cc := collect(simplify(subs(subs(THsets, SC2T_temp5cc))), B[1], diff, ATc): 
> # Solvability condition - Hopf 
> SC2H_temp1 := PDETools[dsubs](\[seq(u_{i1}(vas)=ans1[i], i=1..sys_dim)\], SC2H): 
> SC2H_temp2 := simplify(expand(SC2H_temp1)): 
> SC2H_temp3 := coeff(SC2H_temp2, exp(I*w*T[0]0)): 
> SC2H_temp4 := collect(simplify(SC2H_temp3), B[1], diff, 'distributed'): 
> SC2H_temp5 := collect( subs(THsets, SC2H_temp4), B[1], diff, AH): 
> SC2H_temp6 := denom(simplify(SC2H_temp5)) * simplify(SC2H_temp5) = 0: 
> SC2H_temp6cc := collect(simplify(subs(subs(THsets, SC2H_temp6cc))), B[1], diff, AHc): 

Amp_solving yields the order-2 solvability condition:

> SC2T Case;
  AT B[1] \sqrt{N[0]} + (-A - A^2 \sqrt{N[0]}) AT N[1] + 
  (N[0]^{3/2} + N[0]^2 A - \sqrt{N[0]} - N[0] A) \frac{\partial}{\partial T[1]} AT = 0

> SC2H Case;

The order-2 solvability condition gives rise to following constraints:

\[
\frac{\partial}{\partial T[1]} AT = \frac{\partial}{\partial T[1]} A_T^* = \frac{\partial}{\partial T[1]} AH = \frac{\partial}{\partial T[1]} A_H = 0
\]

\[
B[1] = 0, \sigma_1 = 0
\]

which are defined in Amp_solving as:

> aas := R[1], R[2], T[2], R[3], T[3]: 
> B[1] := 0: 
> N[1] := 0:

Using the order-2 solvability condition, we can solve the order-2 equation.

> for j from 1 to sys_dim do 
  eq||j||_order2_withAnsatz_temp := 
  PDETools[dsubs](\[seq(u_{i1}(vas)=ans1[i], i=1..sys_dim)\], eq||j||_order2_diff): 
  eq||j||_order2_withAnsatz :=
Finally, apply the solvability condition Eqs. (39a) and (39b) to the order-3 equation to derive the order-3 amplitude equations for the Turing and Hopf components.
TH mode amplitude equations are consistent with those derived by de Wit [1993].

\[ \text{Eq. (24)}, \text{ and Eq. (C.3) becomes the amplitude equation for the pure Hopf mode.} \]

Removing cross terms, Eq. (C.1) becomes the amplitude equation for the stripes mode (i.e., Eq. (24), and Eq. (C.3) becomes the amplitude equation for the pure Hopf mode. Amp_solving derived TH mode amplitude equations are consistent with those derived by de Wit [1993].
### Appendix D Symbol notations in Amp.solving

<table>
<thead>
<tr>
<th>Mathematical symbol</th>
<th>Amp.solving expression</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\partial/\partial t$</td>
<td>Dt()</td>
<td>partial derivative in time</td>
</tr>
<tr>
<td>$\nabla$</td>
<td>Ds()</td>
<td>partial derivative in space</td>
</tr>
<tr>
<td>$\nabla^2$</td>
<td>DsDd2()</td>
<td>second spatial derivative (Laplacian)</td>
</tr>
<tr>
<td>$X, Y$</td>
<td>u0, u2,0</td>
<td>Brusselator variables</td>
</tr>
<tr>
<td>$X_0, Y_0$</td>
<td></td>
<td>Brusselator steady-states</td>
</tr>
<tr>
<td>$\eta$</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>$\sigma$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td>Bo</td>
<td>bifurcation parameter</td>
</tr>
<tr>
<td>$\delta x, \delta y$</td>
<td>du1, du2</td>
<td>linear perturbations</td>
</tr>
<tr>
<td>$x_1, x_2, x_3$</td>
<td>u1, u2, u3</td>
<td>expanded variables for $\delta x$</td>
</tr>
<tr>
<td>$y_1, y_2, y_3$</td>
<td>u21, u22, u23</td>
<td>expanded variables for $\delta y$</td>
</tr>
<tr>
<td>$B_0, B_1, B_2$</td>
<td>B0, B1, B2</td>
<td>expanded $B$</td>
</tr>
<tr>
<td>$\sigma_0, \sigma_1, \sigma_2$</td>
<td>N0, N1, N2</td>
<td>expanded $\sigma$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>e</td>
<td>scaling constant</td>
</tr>
<tr>
<td>$I_j$</td>
<td>eq_vec_orderj</td>
<td></td>
</tr>
<tr>
<td>$R_T$</td>
<td>RT</td>
<td>Turing critical right eigenvectors</td>
</tr>
<tr>
<td>$L_T$</td>
<td>LT</td>
<td>Turing critical left eigenvectors</td>
</tr>
<tr>
<td>$R_H$</td>
<td>RH</td>
<td>Hopf critical right eigenvectors</td>
</tr>
<tr>
<td>$R_H^\ast$</td>
<td>RHc</td>
<td>Hopf critical right eigenvectors (c.c.)</td>
</tr>
<tr>
<td>$L_H$</td>
<td>LH</td>
<td>Hopf critical left eigenvectors</td>
</tr>
<tr>
<td>$L_H^\ast$</td>
<td>LHc</td>
<td>Hopf critical left eigenvectors (c.c.)</td>
</tr>
<tr>
<td>$q_c$</td>
<td>q0</td>
<td>Turing critical wavenumber</td>
</tr>
<tr>
<td>$\omega$</td>
<td>w</td>
<td>Hopf critical frequency</td>
</tr>
<tr>
<td>$A_{T1}, A_{T2}, A_{T3}$</td>
<td>AT1, AT2, AT3</td>
<td>hexagonal amplitudes</td>
</tr>
<tr>
<td>$A_{H1}, A_{T2}, A_{T3}$</td>
<td>AT1c, AT2c, AT3c</td>
<td>hexagonal amplitudes (c.c.)</td>
</tr>
<tr>
<td>$Z_1, Z_2, Z_3$</td>
<td>Z1, Z2, Z3</td>
<td></td>
</tr>
<tr>
<td>$q_1, q_2, q_3$</td>
<td>q1, q2, q3</td>
<td>hexagonal wavevectors</td>
</tr>
<tr>
<td>$A_T, A_H$</td>
<td>AT, AH</td>
<td>Turing, Hopf amplitudes for the TH mode</td>
</tr>
<tr>
<td>$A_T^\ast, A_H^\ast$</td>
<td>ATc, AHc</td>
<td>Turing, Hopf amplitudes for the TH mode (c.c.)</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>c1_j</td>
<td>coefficients in the order-2 solution</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>$i$</td>
<td>1</td>
<td>$\sqrt{-1}$</td>
</tr>
</tbody>
</table>

**References**


