Symbolic Computation for Nonlinear Wave Resonances

E. Kartashova, C. Raab, Ch. Feurer, G. Mayrhofer, and W. Schreiner

Abstract Extreme ocean waves are characterized by the energy concentration in a few chosen waves/modes. Frequency modulation due to the nonlinear resonances is one of the possible processes yielding the appearance of independent wave clusters which keep their energy. Energetic behavior of these clusters is defined by (1) integer solutions of the resonance conditions, and (2) coupling coefficients of the dynamical system on the wave amplitudes. General computation algorithms are presented which can be used for arbitrary 3-wave resonant system. Implementation in Mathematica is given for planetary ocean waves. Short discussion concludes the paper.

1 Introduction

Resonance is a common thread that runs through almost every branch of physics, and without resonance we would not have radio, television, music, etc. Resonance causes an object to oscillate, sometimes the oscillation is easy to see (vibration in a guitar string), but sometimes this is impossible without measuring instruments (electrons in an electrical circuit). A well-known example with Tacoma Narrows Bridge (at the time it opened for traffic in 1940, it was the third longest suspension bridge in the world) shows how disastrous resonances can be: on the morning of 7 November 1940, the 4-month-old Tacoma Narrows Bridge began to oscillate dangerously up and down, tore itself apart, and collapsed. Though designed for winds of 120 mph, a wind of only 42 mph caused it to collapse. The experts did agree that somehow the wind caused the bridge to resonate, and nowadays, wind tunnel testing of bridge designs is mandatory.

Another famous example is the experiments of Tesla who in 1898 studied experimentally vibrations of an iron column and noticed that at certain frequencies specific

E. Kartashova, C. Raab, Ch. Feurer, G. Mayrhofer, and W. Schreiner
Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria

pieces of equipment in the room would start to jiggle. Playing with the frequency he was able to move the jiggle to another part of the room. Completely fascinated with these findings, he forgot that the column ran downward into the foundation of the building, and the vibrations were being transmitted all over Manhattan. The experiments had started sort of a small earthquake in his neighborhood with smashed windows, swayed buildings, and panicky people in the streets. For Tesla, the first hint of trouble came when the walls and floor began to heave (Cheney 1989). He stopped the experiment as soon as he saw police rushing through the door.

The difference between resonances in a human made system and in some natural phenomena is very simple. We can change the form of a bridge and stop the experiment by switching off electricity but we can not change the direction of the wind, the form of the Earth atmosphere, or the sizes of an ocean. What we can try to do is *to predict* drastic behavior of a real physical system by computing its resonances. While linear resonances in different physical systems are comparatively well studied, to compute characteristics of nonlinear resonances and to predict their properties is quite a nontrivial problem, even in the one-dimensional case. Thus, the notorious Fermi–Pasta–Ulam numerical experiments with a nonlinear *1D*-string (carried out more than 50 years ago) are still not fully understood (Berman and Israilev 2005). On the other hand, nonlinear wave resonances in continuous *2D*-media like ocean, space, atmosphere, plasma, etc. are well studied in the frame of wave turbulence theory (Zakharov et al. 1992) and provide a sound basis for qualitative and sometimes also quantitative analysis of corresponding physical systems. The notion of nonlinear wave interactions is crucial in the wave turbulence theory (Zakharov et al. 2004). Excluding resonances allows to describe a nonlinear wave system statistically by wave kinetic equations and power-law energy spectra of turbulence (Zakharov and Filonenko 1967), and to observe this behavior in numerical experiments (Pushkarev and Zakharov 2000). Direct computations with Euler equations (modified for gravity water waves (Zakharov et al. 2005)) show that the existence of resonances in a wave system yield some additional effects, which are not covered by the statistical description. The role of resonances in the evolution of water wave turbulent systems has been studied profoundly by a great number of researchers. One of the most important conclusions (for gravity water waves) made recently in Tanaka (2007) is the following: *The four-wave resonant interactions control the evolution of the spectrum at every instant of time, whereas nonresonant interactions do not make any significant contribution even in a short-term evolution*.

The behavior of a resonant wave system can be briefly described (Kartashova 1998) as follows: (1) not all waves take part in resonant interactions, (2) resonantly interacting waves form a few independent small wave clusters, such that there is no energy flow between these clusters, (3) including some small but nonzero resonance width into consideration *does not* destroy the clusters. A model of laminated wave turbulence (Kartashova 2006a) allows to describe statistical and resonant regimes simultaneously while methods to compute resonances numerically are presented in Kartashova (2006b) (idea) and in Kartashova and Kartashov (2006, 2007a,b) (implementation). Our main purpose here is to study the possibilities of a symbolic implementation of these general algorithms using the computer algebra system Mathematica.

The implemented software can be executed with local installations of Mathematica and the corresponding method libraries; however, we have also developed a Web interface that allows to run the methods from any computer in the Internet via a conventional Web browser. The implementation strategy is simple and is based on generally available technologies; it can serve as a blueprint for other mathematical software with similar features.

We take as our principal example the barotropic vorticity equation in a rectangular domain with zero boundary conditions, which describes oceanic planetary waves, and show how (a) to compute interaction coefficients of corresponding dynamical systems, (b) to solve resonant conditions, (c) to construct the topological structure of the solution set, and (d) to use the software via a Web interface over the Internet. A short discussion concludes the paper.

2 Mathematical Background

Wave turbulence takes place in physical systems with nonlinear dispersive waves that are described by evolutionary dispersive NPDEs. The role of the evolutionary dispersive NPDEs in the theoretical physics is so important that the notion of dispersion is used for *a physical* classification of PDEs into dispersive and nondispersive. On the one hand, the well-known mathematical classification of PDEs into elliptic, parabolic, and hyperbolic equations is based on the form of equations and can be applied to the second order PDEs on an arbitrary number of variables. On the other hand, the physical classification is based on *the form of solutions* and can be applied to PDEs of arbitrary order and arbitrary number of variables. To construct the physical classification of PDEs, two preliminary steps are to be made: (1) to divide all variables into two groups – time- and space-like variables (t and x, respectively); and (2) to check that the *linear part* of the PDE under consideration has a wave-like solution in the form of Fourier harmonic

$$\psi(x,t) = A \exp \mathrm{i}[kx - \omega t],$$

with amplitude A, wave-number k, and wave frequency ω. The direct substitution of this solution into the linear PDE shows that ω is an explicit function on k, for instance:

$$\psi_t + \psi_x + \psi_{xxx} = 0 \quad \Rightarrow \quad \omega(k) = k - 5k^3.$$

If ω as a function on k is real-valued and such that $\mathrm{d}^2\omega/\mathrm{d}k^2 \neq 0$, it is called *a dispersion function* and the corresponding PDE is called evolutionary dispersive PDE. If the dimension of the space variable x is more than 1, that is, $\vec{x} = (x_1, \dots, x_p)$, $\vec{k}$ is called the wave-vector and the dispersion function $\omega = \omega(\vec{k})$ depends on the coordinates of the wave-vector. This classification *is not complementary* to a standard mathematical one. For instance, though hyperbolic PDEs normally do not have dispersive wave solutions, the hyperbolic equation $\psi_{tt} - \alpha^2 \psi_{xx} - \beta^2 \psi = 0$ has them.

In the huge amount of application areas of NPDEs (classical and quantum physics, chemistry, medicine, sociology, etc.), a nonlinear term of the corresponding NPDE can be regarded as small. This is symbolically written as

$$L(\psi) = -\varepsilon N(\psi), \tag{1}$$

where L and N are linear and nonlinear parts of the equation respectively and ε is a small parameter defined explicitly by the physical problem setting. It can be shown that in this case the solution ψ of (1) can be constructed as a combination of the Fourier harmonics with amplitudes A depending on the time variable and possessing two properties formulated here for the case of quadratic nonlinearity:

- **P1.** The amplitudes of the Fourier harmonics satisfy the following system of nonlinear ordinary differential equations (ODEs) written for simplicity in the real form

$$\begin{aligned} \dot{A}_1 &= \alpha_1 A_2 A_3, \\ \dot{A}_2 &= \alpha_2 A_1 A_3, \\ \dot{A}_3 &= \alpha_3 A_1 A_2, \end{aligned} \tag{2}$$

  with coefficients α_j being functions on wave-numbers;
- **P2.** The dispersion function and wave-numbers satisfy the *resonance conditions*

$$\begin{cases} \omega(\vec{k}_1) \pm \omega(\vec{k}_2) \pm \omega(\vec{k}_3) = 0, \\ \vec{k}_1 \pm \vec{k}_2 \pm \vec{k}_3 = 0. \end{cases} \tag{3}$$

The transition from (1) to (2) can be performed by some standard methods (for instance, multiscale method (Nayfeh 1981)), which also yields the explicit form of resonance conditions.

Keeping in mind our main problem – to find a solution of (1) – one has to take care of the initial and boundary conditions. This is done in the following way: the case of periodic or zero boundary conditions yields *integer wave numbers*, otherwise they are real. Correspondingly, one has to find all integer (or real) solutions of (3), substitute corresponding wave-numbers into the coefficients α_j, and then look for the solutions of (2) with given initial conditions.

One can see immediately a big problem that appears as soon as one has to solve a NPDE with periodical or zero boundary conditions. Indeed, dispersion functions take different forms, for instance,

$$\omega^2 = k^3,\ \ \omega^2 = k^3 + \alpha k,\ \ \omega^2 = k,\ \ \omega = \alpha/k,\ \ \omega = m/n(n+1), \ldots, \text{etc.},$$

with $\vec{k} = (m,n)$, $k = \sqrt{m^2+n^2}$, and α being a constant. This means that (3) corresponds to a system of Diophantine equations of many variables, normally 6–9, with cumulative degrees 10–16. Those have to be solved usually for the integers of the order $\sim 10^3$, which means that computations has to be performed with integers of order 10^{48} and more. Original algorithms to solve these systems of equations have been developed based on some profound results of number theory (Kartashova 2006b) and implemented numerically (Kartashova and Kartashov 2006, 2007a,b).

Further on, an evolutionary dispersive NPDE with periodic or zero boundary conditions is called *three-term mesoscopic system* if it has a solution of the form

$$\tilde{\psi} = \sum_{j=1}^{\infty} A_j \exp \mathrm{i}[\vec{k}_j \vec{x}_j - \omega t]$$

and there exists at least one triple $\{A_{j_1}, A_{j_2}, A_{j_3}\} \in \{A_j\}$ such that **P1** and **P2** keep true with some nonzero coefficients $\alpha_j,\ \alpha_j \neq 0\ \forall j = 1,2,3.$

3 Equations for Wave Amplitudes

3.1 Method Description

The barotropic vorticity equation describing ocean planetary waves has the form (Kartashova and Reznik 1992)

$$\frac{\partial \triangle \psi}{\partial t} + \beta \frac{\partial \psi}{\partial x} = -\varepsilon J(\psi, \triangle \psi), \tag{4}$$

with boundary conditions

$$\psi = 0 \quad \text{for } x = 0,\ L_x;\ \ y = 0,\ L_y.$$

Here β is a constant called Rossby number, ε is a small parameter and the Jacobian has the standard form

$$J(a,b) = \frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial a}{\partial y}\frac{\partial b}{\partial x}.$$

First we give a basic introduction on how a PDE can be turned into a system of ODEs by a multiscale method. Using operator notation, our problem (4) is viewed as a perturbed version of the linear PDE $L(\psi) = 0$. We pick a solution of this equation, say ψ_0, which is a superposition of several waves φ_j, that is, $\psi_0 = \sum_{j=1}^{s} A_j \varphi_j$, each being a solution itself. To construct a solution of the original problem we make the amplitudes time-dependent. As the size of the nonlinearity in (1) is just of order ε, the amplitudes will vary only on time-scales $1/\varepsilon$ times slower than the waves. Hence we define an additional time-variable $t_1 := t\varepsilon$ called "slow time" to handle this time scale. So we look for approximate solutions of (1) that have the following form

$$\psi_0(t, t_1, \vec{x}) = \sum_{j=1}^{s} A_j(t_1) \varphi_j(\vec{x}, t),$$

which for $\varepsilon = 0$ is an exact solution. The exact solution of the equation is written as power series in ε around ψ_0, that is, $\psi = \sum_{k=0}^{\infty} \psi_k \varepsilon^k$. In our computation it is truncated up to maximal order m, which in our case is $m = 1$, that is,

$$\psi(t,t_1,\vec{x}) = \psi_0(t,t_1,\vec{x}) + \psi_1(t,t_1,\vec{x})\varepsilon.$$

Plugging $\psi(t,t_1,\vec{x})$ one has to keep in mind that, since $t_1 = \varepsilon t$, we now have $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \varepsilon \frac{\partial}{\partial t_1}$ due to the chain rule. Equations are formed by comparing the coefficients of ε^k. For $k = 0$ this gives back the linear equation, but we keep the equation for $k = 1$. In particular, for (4) we arrive at

$$\begin{aligned} \frac{\partial \triangle \psi_0}{\partial t} + \beta \frac{\partial \psi_0}{\partial x} &= 0, \\ \frac{\partial \triangle \psi_0}{\partial t_1} + \frac{\partial \triangle \psi_1}{\partial t} + \beta \frac{\partial \psi_1}{\partial x} &= -J(\psi_0, \triangle \psi_0). \end{aligned}$$

To get (2), we have to get rid of all other variables. This is done by integrating against the φ_j's, that is, $\langle ., \varphi_j \rangle_{L^2(\Omega)}$, and averaging over (fast) time, that is, $\lim_{T\to\infty} \frac{1}{T} \int_0^T . \,\mathrm{d}t$.

3.2 The Implementation

This method was implemented in Mathematica with order $m = 1$ in mind. So it would not be immediately applicable to higher orders without some (minor) adjustments. The ODEs are constructed by the function

```
ODESystem[L(ψ), N(ψ), ψ,
    {x1,..,xn}, t, domain, jacobian, m, s, A, linwav,
    {λ1,..,λp}, paramvalues].
```

Basically this function takes the problem together with the solution of the linear equation as input and computes the list of ODEs for the amplitudes as output. Its arguments are given in more detail:

- `L(ψ)`,`N(ψ)`: Linear and nonlinear part of (1), each applied to a symbolic function parameter. Derivatives have to be specified with `Dt` instead of `D` and the nonlinear part has to be a polynomial in the derivatives of the function.
- ψ: symbol used for function in `L(ψ), N(ψ)`
- `{x1,...,xn}, t`: list of symbols used for space-variables, and symbol for time-variable
- `domain`: The domain on which the equation is considered has to be specified in the form `{{x1,minx1,maxx1}, ..., {xn,minxn,maxxn}}`, where the bounds on $\mathtt{x}_i$ may depend on `x1,...,x`$_{i-1}$ only.
- `jacobian`: For integration the (determinant of the) Jacobian must also be passed to the function. This is needed in case the physical domain does not coincide with the domain of the variables above, it can be set to `1` otherwise.

- `m, s`: maximal power of ε and number of waves considered
- `A`: symbol used for amplitudes
- `linwav`: General wave of the linear equation is assumed to have separated variables, that is, $\varphi(\vec{x},t) = B_1(x_1)\cdots B_n(x_n)\,\exp(\mathrm{i}\theta(x_1,\ldots,x_n,t))$, and has to be given in the form
  $\{$`B`$_1$`(x`$_1$`),...,B`$_n$`(x`$_n$`),` θ`(x`$_1$`,...,x`$_n$`,t)`$\}$.
- $\{\lambda_1,\ldots,\lambda_p\}$: list of symbols of parameters the functions in `linwav` depend on
- `paramvalues`: For each of the `s` waves explicit values of the parameters $\{\lambda_1,\ldots,\lambda_p\}$ have to be passed as a list of `s` vectors of parameter values.

```
ODESystem[linearpart_,nonlinearpart_,fun_Symbol,vars_List,
    t_Symbol,domain_List,jacobian_,ord_Integer,num_Integer,
    A_Symbol,linwav_List,params_List,paramvalues_List] :=
  Module[{B,theta,eq,k},
    eq = PerturbationEqns[linearpart,nonlinearpart,
                          fun,vars,t,ord];
    eq = PlugInGenericWaveTuple[eq,fun,vars,t,A,B,theta,num]
             /. fun[1]->(0&);
    eq = Table[Resonance2[eq,linwav,vars,t,params,A,B,theta,
                   num,paramvalues,k],
               {k,num}];
    Map[Integrate[Simplify[#,And@@(Function[B,B[[2]]<B[[1]]<
                     B[[3]]]/@domain)]*jacobian,
                 Sequence@@domain]&,
        eq,{2}]
    ]
```

Internally this function is divided into three subroutines briefly described below.

3.2.1 Perturbation Equations, General Form

The first of the subroutines is

`PerturbationEqns[L(`ψ`), N(`ψ`),` ψ`, {x`$_1$`,...,x`$_n$`}, t, m]`.

As mentioned before we approximate the solution of our problem by a polynomial of degree m in ε. This subroutine works for arbitrary m. In the first step we construct equations by coefficient comparison. Additional time-variables will be created automatically and labeled `t[1],...,t[m]`. The output is a list of $m+1$ equations corresponding to the powers $\varepsilon^0,\ldots,\varepsilon^m$. The implementation is quite straightforward. First set $\psi = \sum_{k=0}^{m}\psi_k(t,t_1,\ldots,t_m,x_1,\ldots,x_n)\varepsilon^k$ in (1), where $t_k = \varepsilon^k t$, that is, $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \sum_{k=1}^{m}\varepsilon^k\frac{\partial}{\partial t_k}$. Then extract the coefficients of $\varepsilon^0,\ldots,\varepsilon^m$ on both sides and assemble the equations. Finally replace $\varepsilon^k t$ by t_k again.

```
PerturbationEqns[linearpart_,nonlinearpart_,fun_Symbol,
    vars_List,time_Symbol,ord_Integer] :=
  Module[{i,j,e,eq},
    eq = ((linearpart == -e*nonlinearpart)
            /. {fun->Sum[e^i*fun[i][time,Sequence@@Table[e^j*
                                    time,{j,ord}],Sequence@@
                                    DeleteCases[vars,time]],
                          {i,0,ord}]});
    eq = (eq /. ((Dt[#, __]->0)& /@ Join[vars,{time,e}]));
    eq = (Equal@@#)& /@
             Transpose[Take[CoefficientList[#,e],1+ord]& /@
                          (List@@eq)];
    eq /. Table[e^j*time->time[j],{j,ord}]
    ]
```

3.2.2 Perturbation Equations, Given Linear Mode

In step two we set $\psi_0(t,t_1,\vec{x}) = \sum_{j=1}^{s} A_j(t_1)\varphi_j(\vec{x},t)$ as described earlier. This is done by the function

`PlugInGenericWaveTuple[eq,` ψ, $\{x_1,\ldots,x_n\}$, `t, A, B,` θ, `s],`

where the first argument is the output of the previous step. The symbols `B` and θ have to be passed for labeling the shape and phase functions, respectively. The output consists of two parts. The first part of the list formulates the assumption $L(\varphi_j) = 0$ explicitly for each of the waves. This is not used in subsequent computations, but is provided as a way to check the assumption. The second part of the list is the equation corresponding to the coefficients of ε from the previous step, with ψ_0 as above. As the task of this step is so short, the implementation does not need further explanation.

```
PlugInGenericWaveTuple[eq_List,fun_Symbol,vars_List,
  t_Symbol,A_Symbol,B_Symbol,theta_Symbol,num_Integer] :=
  Module[{i,j,waves,n=Length[DeleteCases[vars,t]]},
    waves = Table[A[j][Slot[2]]*
                    Product[B[i][j][Slot[i+2]],{i,n}]*
                    Exp[I*theta[j][Sequence@@Table[Slot[i+2],
                                          {i,n}],Slot[1]]],
                  {j,num}];
    {Table[eq[[1]] /. fun[0]->Function[Evaluate[waves[[j]]]],
           {j,num}],
     Expand /@
         (eq[[2]] /. fun[0]->Function[Evaluate[Total[waves]]])
    }]
```

3.2.3 Time and Scale Averaging

Step three is the most elaborate. Under the assumption that interchange of averaging over time and inner product is justified, an integrand

$$h = \lim_{T\to\infty} \frac{1}{T} \int_0^T \psi_0 \overline{\varphi_k}\, \mathrm{d}t$$

is computed, which when integrated over the domain yields

$$\int_\Omega h = \lim_{T\to\infty} \frac{1}{T} \int_0^T \langle \psi_0, \varphi_k \rangle_{L^2(\Omega)}\, \mathrm{d}t.$$

Resonance conditions posed on the phase functions are explicitly used by

```
Resonance[eq, linwav, {x_1,..,x_n}, t,
    {λ_1,..,λ_p}, A, B, θ, s, cond, k],
```

which receives the output from the previous step in eq. Here cond specifies the resonance condition in terms of θ_j, which have to be entered as θ[j][x_1,..,x_n,t], respectively. The last argument is the index of the wave φ_k in the integral above. Alternatively, Resonance2 uses explicit parameter settings paramvalues for the waves instead of cond. This has been necessary because the general Resonance does not give useable results (see Sect. 3.3 for more details). The main work in this step is to find out which terms do not contribute to the result. We exploit the fact that oscillating terms vanish when averaged over time by simply omitting those summands of $\langle \psi_0, \varphi_k \rangle_{L^2(\Omega)}$ that have a factor $\exp(\mathrm{i}\theta)$ with some time-dependent phase θ. The code for Resonance is not shown here, but is quite similar to Resonance2.

```
Resonance2[eq_List,linwav_List,vars_List,t_Symbol,params_List,
    A_Symbol,B_Symbol,theta_Symbol,num_Integer,
    paramvalues_List,testwave_Integer] :=
  Module[{e,i,j,n=Length[DeleteCases[vars,t]]},
    e = Expand[(List@@Last[eq])*
             Exp[-I*theta[testwave][Sequence@@
                                        DeleteCases[vars,t],
                                      t]]];
    e = e /.
         Table[
           theta[j] ->
            (Evaluate[(linwav[[n+1]] /.
                        (Rule@@#& /@
                          Transpose[{params,paramvalues[[j]]}]
                        )
                      ) /. Append[Table[
                                 DeleteCases[vars,t][[i]]
                                   -> Slot[i],
                                 {i,n}],
                               t -> Slot[n+1]]
                     ]&
            ),
          {j,num}];
    e = MapAt[
          (Function[theta,If[FreeQ[theta,t],theta,0]
                    ]
               [Simplify[#]]
          )&,
          e,
          Position[e,Exp[_]]];
    e = Equal@@
          (e*Conjugate[A[testwave]][t[1]]*
            Product[Conjugate[B[i]
                                [testwave]
                                [DeleteCases[vars,t][[i]]]
                              ],
                     {i,n}]
          ) /.
            Flatten[
              Table[B[i][j] ->
                     Function[
                       Evaluate[DeleteCases[vars,t][[i]]],
                       Evaluate[linwav[[i]] /.
                                  (Rule@@#& /@
                                 Transpose[
                                   {params,paramvalues[[j]]
                                   }]
                               )]],
                    {i,n},{j,num}]]
    ]
```

The integration of *h* is done by Mathematica and can be quite time-consuming. So `ODESystem` simplifies the integrand first to make integration faster. Still the expressions involved can be quite complicated. This is the most time-consuming part during construction of the ODEs.

3.3 Obstacles

Mathematica sometimes does not seem to take care of special cases and consequently has problems with evaluating expressions depending on symbolic parameters. We give two simple examples to illustrate this issue:

- Orthogonality of sine-functions.
  Indeed, it holds that

$$\forall m,n \in \mathbb{N}: \int_0^{2\pi} \sin(mx)\sin(nx)\,\mathrm{d}x = \pi\delta_{m,n}.$$

Computing this in Mathematica by

```
Integrate[Sin[m*x]Sin[n*x], {x,0,2π},
    Assumptions → m∈Integers && n∈Integers]
```

yields 0 independently of m,n instead.

- Computation of a limit.
  Mathematica evaluates an expression

$$\forall n \in \mathbb{Z} : \lim_{x\to n} \frac{\sin(x\pi)}{x} = \pi\delta_{n,0}$$

and similar expressions in two different ways getting two different answers. On the one hand

```
Limit[Sin[(m-n)π]/(m-n), m→n,
Assumptions → m∈Integers && n∈Integers]
```

gives 0. On the other hand, however, when the condition $m,n \in \mathbb{Z}$ is not used for computing the result, Mathematica yields the correct answer π, as with

```
Limit[Sin[(m-n)π]/(m-n), m→n].
```

Unfortunately these issues prevented us from obtaining a nice formula for the coefficients in symbolic form by `Resonance`. So we just compute results for explicit parameter settings using `Resonance2`.

3.4 Results

3.4.1 Atmospheric Planetary Waves

For the validation of our program we consider the barotropic vorticity equation on the sphere first. Here numerical values of the coefficients α_i are available (Table 1, Kartashova and L'vov (2007)). The equation looks quite similar

$$\frac{\partial \triangle \psi}{\partial t} + 2\frac{\partial \psi}{\partial \lambda} = -\varepsilon J(\psi, \triangle \psi).$$

However, in spherical coordinates ($\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, $\lambda \in [0, 2\pi]$) the differential operators are different:

$$\triangle = \frac{\partial^2}{\partial \phi^2} + \frac{1}{\cos(\phi)^2}\frac{\partial^2}{\partial \lambda^2} - \tan(\phi)\frac{\partial}{\partial \phi},$$

$$J(a,b) = \frac{1}{\cos(\phi)}\left(\frac{\partial a}{\partial \lambda}\frac{\partial b}{\partial \phi} - \frac{\partial a}{\partial \phi}\frac{\partial b}{\partial \lambda}\right).$$

The linear modes have in this case the following form (Pedlosky 1987):

$$P_n^m(\sin(\phi))\exp\left(\mathrm{i}\left(m\lambda+\frac{2m}{n(n+1)}t\right)\right), \tag{5}$$

where $P_n^m(\mu)$ are the associated Legendre polynomials of degree n and order $m \le n$, so again they depend on the two parameters m and n. Also resonance conditions on the parameters look different in this case.

Now we compute the coefficient α_3 in (2). In Kartashova and L'vov (2007) we find the following equation for the amplitude A_3

$$n_3(n_3+1)\frac{\partial A_3}{\partial t_1}(t_1) = 2iZ(n_2(n_2+1)-n_1(n_1+1))A_1(t_1)A_2(t_1),$$

so $\alpha_3 = 2iZ\frac{n_2(n_2+1)-n_1(n_1+1)}{n_3(n_3+1)}$. Parameter settings and corresponding numerical values for Z were taken from the table below (see Kartashova and L'vov (2007)). For this equation and $s = 3$ results produced by our program have the form $c_1\overline{A_3}\dot{A_3} = c_2A_1A_2\overline{A_3}$, so $\alpha_3 = c_2/c_1$.

Testing all resonant triads from Table 1 from Kartashova and L'vov (2007), we see that the coefficients differ merely by a constant factor of $\pm\sqrt{8}$, which is due to the different scaling of the Legendre polynomials. In our computation they were normalized s.t. $\int_{-1}^{1} P_n^m(\mu)^2\,\mathrm{d}\mu = 1$. With three triads, however, results were completely different. Interestingly this were exactly those triads for which no φ_0 appears in the table.

Furthermore, for the other coefficients in (2) our program computes $\alpha_1 = \alpha_2 = 0$ in all tested parameter settings. This fact can be easily understood in the following way. We checked only resonance conditions but not the conditions for the interaction coefficients to be non-zero, which are elaborated enough:

$$m_i \le n_i, \quad n_i \ne n_j \quad \forall i = 1,2,3, \quad |n_1 - n_2| < n_3 < n_1 + n_2,$$

and

$$n_1 + n_2 + n_3 \quad \text{is odd.}$$

Randomly taken parameter setting does not satisfy these conditions.

3.4.2 Ocean Planetary Waves

Returning to the original example on the domain $[0, L_x] \times [0, L_y]$, we find explicit formulae for the coefficients in Kartashova and Reznik (1992). According to Sect. 3.3 we can only verify special instances and not general formulae.

Linear modes have now the form (Kartashova and Reznik 1992)

$$\sin\left(\pi\frac{mx}{L_x}\right)\sin\left(\pi\frac{ny}{L_y}\right)\exp\left(i\left(\frac{\beta}{2\omega}x+\omega t\right)\right), \tag{6}$$

with $m, n \in \mathbb{N}$ and $\omega = \frac{\beta}{2\pi\sqrt{(\frac{m}{L_x})^2+(\frac{n}{L_y})^2}}$.

Parameter settings solving the resonance conditions were computed as in Sect. 4. Unfortunately results do not match and we have no explanation for that. In particular, the condition $\frac{\alpha_1}{\omega_1^2}+\frac{\alpha_2}{\omega_2^2}+\frac{\alpha_3}{\omega_3^2}=0$ stated in Kartashova and Reznik (1992) does not hold for the results of our program since we got $\alpha_1=\alpha_2=0$ in all tested parameter settings, just as in the spherical case.

For example, if we try the triad `{{2,4},{4,2},{1,2}}`, where $L_x=L_y=1$, our program computes $\alpha_3=\frac{32\sqrt{5}}{11}\pi\left(\sin(3\sqrt{5}\pi)-\mathrm{i}(1+\cos(3\sqrt{5}\pi))\right)$, whereas the general formula yields $\alpha_3=\frac{19+7\sqrt{5}}{11}\pi\sin(3\sqrt{5}\pi)$. However, if we use a triad with $q=1$, e.g. `{{24,18},{9,12},{8,6}}`, both agree on $\alpha_1=\alpha_2=\alpha_3=0$.

4 Resonance Conditions

The main equation to solve is

$$\frac{1}{\sqrt{\left(\frac{m_1}{L_x}\right)^2+\left(\frac{n_1}{L_y}\right)^2}}+\frac{1}{\sqrt{\left(\frac{m_2}{L_x}\right)^2+\left(\frac{n_2}{L_y}\right)^2}}=\frac{1}{\sqrt{\left(\frac{m_3}{L_x}\right)^2+\left(\frac{n_3}{L_y}\right)^2}}$$

for all possible $m_i,n_i\in\mathbb{Z}$ with the scales L_x and L_y (also $\in\mathbb{Z}$) and then to check the condition $n_1\pm n_2=n_3$. In the following argumentation it will be seen that L_x and L_y can be assumed to be free of common factors. Below we refer to L_x and L_y as to *the scale coefficients*.

The first step of the algorithm implemented in Mathematica is to rewrite the equation to $\frac{1}{\sqrt{\tilde{m}_1{}^2+\tilde{n}_1{}^2}}+\frac{1}{\sqrt{\tilde{m}_2{}^2+\tilde{n}_2{}^2}}=\frac{1}{\sqrt{\tilde{m}_3{}^2+\tilde{n}_3{}^2}}$ and transform it in the following way: we factorize the result of each $\tilde{m}_i{}^2+\tilde{n}_i{}^2$ and obtain with $\rho_1\cdots\rho_r$ being the factors of $m_i^2+n_i^2$ and $\alpha_1\cdots\alpha_r$ their respective powers:

$$m_i^2+n_i^2=\rho_1^{\alpha_1}\cdot\rho_2^{\alpha_2}\cdots\rho_r^{\alpha_r}.$$

We now define *a weight γ_i of the wave-vector* (m_i,n_i) as the product of the ρ_j's to the quotient of their respective α_j and 2. The weight q_i will be the name of the product of the ρ_j's, which have an odd exponent:

$$\sqrt{m_i^2+n_i^2}=\gamma_i\sqrt{q_i}.$$

Our equation then can be rewritten as

$$\frac{1}{\gamma_1\sqrt{q_1}}+\frac{1}{\gamma_2\sqrt{q_2}}=\frac{1}{\gamma_3\sqrt{q_3}}$$

and one easily sees that the only way for the equation to possibly hold is $q_1=q_2=q_3=q$ (see Kartashova (2006b) for details). Further, we call q *an index* of the corresponding wave-vectors. The set of all wave-vectors with the same index is called *a class of index q* and is denoted as Cl_q. Obviously, the solutions of the resonance conditions are to be searched for with separate classes only.

At this point one can also see that only such scales, L_x and L_y, without common factors are reasonable. If they had a common factor, it would cancel out in the equation.

4.1 Method Description

The following five steps are the main steps of the algorithm:

- *Step 1:* Compute the list of all possible indexes q.

  To compute the list of all indexes q, we use the fact that they have to be square-free and each factor of q has to be different from $3 \mod 4$ (Lagrange theorem). There exist 57 possible indexes in our computational domains $q \leq 300$:

  $$\{1,2,5,10,13,17,26,29,34,37,41,53,58,61,65,73,74,82,85,89,\\ 97,101,106,109,113,122,130,137,145,146,149,157,170,173,178,\\ 181,185,193,194,197,202,205,218,221,226,229,233,241,257,\\ 265,269,274,277,281,290,293,298\}$$

- *Step 2:* Solve the weight equation $\frac{1}{\gamma_1}+\frac{1}{\gamma_2}=\frac{1}{\gamma_3}$.

  For solving the weight equation, we transform it into the equivalent form:

  $$\gamma_3 = \frac{\gamma_1 \gamma_2}{\gamma_1+\gamma_2}. \tag{7}$$

  The solution triples $\{\gamma_1,\gamma_2,\gamma_3\}$ can now be found by the two for-loops over γ_1 and γ_2 up to a certain maximum parameter and γ_3 is then found constructively with formula (7).
- *Step 3:* Compute all possible pairs (m_i,n_i) – if there are any – that satisfy $m_i^2+n_i^2=\gamma_i^2\,q$.

  To compute our initial variables m_i, n_i, we use the Mathematica standard function **SumOfSquareRepresentation[d, x]**, which produces a list of all possible representations of an integer x as a sum of d squares, that is, we can find all possible pairs (a,b) with $d=2$ such that they satisfy $a^2+b^2=x$. Therefore, checking the condition $m_i^2+n_i^2=\gamma_i^2\,q$ is easy.
- *Step 4:* Sort out the solutions $\{m1,n1,m2,n2,m3,n3\}$ that do not fulfill the condition $n1 \pm n2 = n3$.
- *Step 5:* Check if by dividing m_i by L_x and n_i by L_y there are still exist some solutions.

  Last two steps are trivial.

4.2 The Implementation

Our implementation is quite straightforward and the main program is based on four auxiliary functions shown in the following subsections.

4.2.1 List of Indexes

The function **constructqs[max]** produces the list of all possible indexes q up to the parameter *max*. The first (obvious) q's $sol = \{1\}$ is given and the function checks the conditions starting with $n = 2$. Every time n satisfies the conditions, it is appended to the list *sol*. If one condition fails, the next $n = n+1$ is considered and so on until n reaches the parameter *max*. Then the list *sol* is returned:

```
Clear[constructqs];

constructqs[n_, sol_List, max_]; n>max := sol (*6*)
constructqs[n_?SquareFreeQ, sol_List, max_]
:= constructqs[n+1, Append[sol, n], max] (*5*)

constructqs[n_?SquareFreeQ, sol_List, max_];
MemberQ[Mod[PrimeFactorList[n], 4], 3]
:= constructqs[n+1, sol, max] (*4*)

constructqs[n_, sol_List, max_]; !SquareFreeQ[n]
:= constructqs[n+1, sol, max] (*3*)
constructqs[1] := {1} (*2*)

constructqs[max_] := constructqs[3, {1}, max] (*1*)
```

4.2.2 Weight Equation

The function **findγs[γmax]** solves the weight equation in the following way. For a fixed γ_1 and γ_2 running between 1 and γmax, it is checked if γ_3 is an integer. If it is, the triple $\{\gamma_1, \gamma_2, \gamma_3\}$ is added to the list *sol*, which is empty at the initial moment. Once γ_2 reaches γmax, it is set to 1 again and the search starts again with $\gamma_1 = \gamma_1 + 1$. This is done as long as both γ_1 and γ_2 are lower than *max*. Finally, the list *sol* is returned:

```
findγs[γmax_, γ1_, γ2_, sol_List];

γ1 > γmax := (Clear[γ3],sol) (*6*)

findγs[γmax_, γ1_, γ2_, sol_List]; (γ1 ≤ γmax && γ2>γmax &&
IntegerQ[γ3=(γ1γ2)/(γ1+γ2)])
:= findγs[γmax, γ1+1, 1, Append[sol, {γ1, γ2, γ3}]] (*5*)

findγs[γmax_, γ1_, γ2_, sol_List];
(γ1 ≤ γmax && γ2>γmax &&
!IntegerQ[γ3=(γ1γ2)/(γ1+γ2)])
:= findγs[γmax, γ1 + 1, 1, sol] (*4*)

findγs[γmax_, γ1_, γ2_, sol_List];
(γ1 ≤ γmax && γ2 ≤ γmax && IntegerQ[γ3=(γ1γ2)/(γ1+γ2)])
:= findγs[γmax, γ1, γ2+1, Append[sol, {γ1, γ2, γ3}]] (*3*)

findγs[γmax_, γ1_, γ2_, sol_List];
(γ1 ≤ γmax && γ2 ≤ γmax && !IntegerQ[γ3=(γ1γ2)/(γ1+γ2)])
:= findγs[γmax, γ1, γ2 + 1, sol] (*2*)

findγs[γmax_] := findγs[γmax, 1, 1, {}]) (*1*)
```

For **findγs[γmax]** to be executable, the iteration depth of 2^{12} is not sufficient and it was set to ∞.

4.2.3 Linear Condition

The third auxiliary function **makemns** checks whether the linear condition $n_1 \pm n_2 = n_3$ is fulfilled and structures the solution set into a list of pairs $\{\{m_1,n_1\},\{m_2,n_2\},\{m_3,n_3\}\}$:

```
Clear[makemns];

makemns[m1_, n1_, m2_, n2_, m3_, n3_] := {} (*3*)

makemns[m1_, n1_, m2_, n2_, m3_, n3_];
(n1 + n2 == n3 || n1 - n2 == n3) :=
    {{m1, n1}, {m2, n2}, {m3, n3}} (*2*)

makemns[mn1_List, mn2_List, mn3_List] :=
    Cases[Flatten[Table[makemns[mn1[[i,1]], mn1[[i,2]],
       mn2[[j,1]], mn2[[j,2]], mn3[[k,1]], mn3[[k,2]]],
       {i, 1, Length[mn1]}, {j, 1, Length[mn2]},
       {k, 1, Length[mn3]}], 2],
       {{x1_,x2_}, {x3_,x4_}, {x5_,x6_}}] (*1*)
```

The function **makemns** is called three times:

In (*1*) from three lists of arbitrarily many pairs {mi, ni}, a three-dimensional array is made combining entries of the three lists with each other. Each entry calls the same program with the parameters of the current combination of {m1,n1,m2,n2,m3,n3}.

In (*2*) and (*3*) it is decided whether the condition $n1 \pm n2 = n3$ is fulfilled. If it is, a solution {{m1,n1},{m2,n2},{m3,n3}} is written in the array. The table is then flattened to the level 2 in order to have a list of solutions. In the end, all empty lists have to be sorted out, done by the function **Cases**, which keeps only those cases that have the shape {{x1_,x2_},{x3_,x4_},{x5_,x6_}}.

4.2.4 Scale Coefficients

Finally, the function **respectL[sol, Lx, Ly]** divides each component of the solution by the pair $(L_x,\ L_y)$ and sorts out the result if any of the six components does not remain an integer:

```
respectL[sol_List, Lx_, Ly_] :=
    Map[solution[#]&,
       Cases[Map[#/{Lx, Ly}&,
       Map[#[[1]]]&, sol], {2}], {{_Integer, _Integer},
      {_Integer, _Integer}, {_Integer,  _Integer}}]]
```

The function **respectL[sol, Lx, Ly]** gets as an input the list of the form {solution[{{m1,n1},{m2,n2},{m3,n3}}],...} and returns the list of the same form.

4.3 Results

All solutions in the computation domain $m,n \leq 300$ have been found in a few minutes. Notice that computations in the domain $m,n \leq 20$ by direct search without introducing indexes q and classes Cl_q took about 30 min. A direct search in the domain $m,n \leq 30$ has been interrupted after 2 h, since no results were produced.

The number of solutions depends drastically on the scales L_x and L_y, some data are given below (for the domain $m,n \leq 50$):

$(L_x = 1, L_y = 1)$: 76 solutions;
$(L_x = 3, L_y = 1)$: 23 solutions;
$(L_x = 6, L_y = 16)$: 2 solutions;
$(L_x = 5, L_y = 21)$: 2 solutions;
$(L_x = 11, L_y = 29)$: no solutions (search up to 300, for both $qmax$ and γmax).

Interestingly enough, in all tried possibilities, only an odd q yield solutions.

5 Structure of the Solution Set

5.1 Method Description

The graphical way to present 2D-wave resonances suggested in Kartashova (1998) for three-wave interactions is to regard each 2D-vector $\vec{k} = (m,n)$ as a node (m,n) of integer lattice in the spectral space and connect those nodes which construct one solution (triad, quartet, etc.). Having computed already all the solutions of (3) in Sect. 4, now we are interested in the structure of resonances in spectral space. To each node (m,n) we can prescribe an amplitude $A(m,n,t_1)$ whose time evolution can be computed from the dynamical equations obtained in Sect. 3. Thus, solution set of resonance conditions (3) can be thought of as a collection of triangles, some of them are isolated, some form small groups connected by one or two vertices. Corresponding dynamical systems can be reconstructed from the structure of these groups. For instance, a single isolated triangle corresponding to a solution with wave vectors $(m_1,n_1)(m_2,n_2)(m_3,n_3)$ and wave amplitudes $\{(A1,\ A2,\ A3)\}$ corresponds to the following dynamical system:

$$\begin{aligned}\dot{A_1} &= \alpha_1 A_2 A_3,\\ \dot{A_2} &= \alpha_2 A_1 A_3,\\ \dot{A_3} &= \alpha_3 A_1 A_2,\end{aligned}$$

with α_i being functions of all m_i, n_i (see Sect. 3).

If that two triangles share one common vertex $\{(A1,\ A2,\ A3),(A3,\ A4,\ A5)\}$, then the corresponding dynamical system is

$$\begin{aligned}\dot{A_1} &= \alpha_1 A_2 A_3,\\ \dot{A_2} &= \alpha_2 A_1 A_3,\\ \dot{A_3} &= \alpha_{3,1} A_1 A_2 + \alpha_{3,2} A_4 A_5,\\ \dot{A_4} &= \alpha_4 A_3 A_5,\\ \dot{A_5} &= \alpha_5 A_3 A_4.\end{aligned}$$

If two triangles have two vertices in common $\{(A1,\ A2,\ A3),(A2,\ A3,\ A4)\}$, then the dynamical system is quite different:

$$\begin{aligned}\dot{A_1} &= \alpha_1 A_2 A_3,\\ \dot{A_2} &= \alpha_{2,1} A_1 A_3 + \alpha_{2,2} A_3 A_4,\\ \dot{A_3} &= \alpha_{3,1} A_1 A_2 + \alpha_{3,2} A_2 A_4,\\ \dot{A_4} &= \alpha_4 A_2 A_3 = \frac{\alpha_4}{\alpha_1}\dot{A_1}.\end{aligned}$$

Using the fourth equation, the formulae for $\dot{A_2}$ and $\dot{A_3}$ can be simplified to

$$\begin{aligned}\dot{A_4} &= \frac{\alpha_4}{\alpha_1}\dot{A_1} \Rightarrow A_4 = \frac{\alpha_4}{\alpha_1}A_1 + \beta_1,\\ \dot{A_2} &= A_1 A_3 \left(\alpha_{2,1} + \frac{\alpha_{2,2}\alpha_4}{\alpha_1}\right) + \frac{\alpha_4 \beta_1}{\alpha_1},\\ \dot{A_3} &= A_1 A_2 \left(\alpha_{3,1} + \frac{\alpha_{3,2}\alpha_4}{\alpha_1}\right) + \frac{\alpha_4 \beta_1}{\alpha_1}.\end{aligned}$$

This means that *qualitative dynamics* of the three-term mesoscopic system depends *not on the geometrical structure* of the solution set but on its *topological structure*. Constructing the topological structure of the solution set, we do not consider concrete values of the solution but only the way how triangles are connected. In any finite spectral domain we can compute all independent wave clusters and write out corresponding dynamical systems; thus obtaining complete information about energy transfer through the spectrum. Of course, *quantitative* properties of the dynamical systems depend on the specific values of m_i, n_i (for instance, values of interaction coefficients α_i, magnitudes of periods of the energy exchange among the waves belonging to one cluster, etc.).

5.2 Implementation

To construct the topological structure of a given solution set we need first to find all groups of connected triangles. This is done by the following procedure:

```
FindConnectedGroups[triangles_List] :=
 Block[{groups = {}, tr = triangles, newgroup},
   While[Length[tr] > 0,
     {newgroup, tr} =
       FindConnectedTriangles[{First[tr]}, Rest[tr]];
     groups = Append[groups, newgroup];
   ];
   groups
 ];

FindConnectedTriangles[grp_List,triangles_List]:=
 Module[{points,newGrpMember,tr=triangles},
   points=Flatten[Apply[List,grp,2],1];
   newGrpMember=Cases[tr, _[___,#1,___]]&/@points;
   (tr=DeleteCases[tr, _[___,#1,___]])&/@points;
   newGrpMember=Union[Join@@newGrpMember];
   If[Length[newGrpMember]==0,
     {grp,tr},
     newGrpMember=FindConnectedTriangles[newGrpMember,tr];
     {Join[grp,First[newGrpMember]],
      newGrpMember[[2]]}
    ]
 ];
```

The function `FindConnectedGroups` expects a list of triangles as input, and three different types for data structure can be used. The first type is just a list of three pairs, where each pair contains the coordinates of a node, for example, `{{1,2},{3,4},{5,6}}`. An alternative type is like the type before just with another head symbol instead of list, for example,

`Triangle[{1,2},{3,4},{5,6}].`

The function also works for vertex numbers instead of coordinates, for example, `Triangle[1,2,3]`. In every case, the function returns a partition of the input list where all elements of a list are connected and elements of different lists have no connection to each other.

The function `FindConnectedTriangles` is an auxiliary function, which has two parameters. The first list contains all connected triangles. The second list contains all other triangles that are possibly connected to one of the triangles in the first list. The function `FindConnectedTriangles` returns a pair of lists: the first list contains all triangles that are connected to the selected triangles, the second list contains all the remaining.

The input list for `FindConnectedTriangles` is a list of three-element lists. Before we can use the results produced in Sect. 4 as an input we have to transform the data. This can be easily done by

```
TransformSolution[sol_List]:=
  Flatten[Rest/@sol]/.solution[trs:{___List}]->trs.
```

Some remarks on the implementation

The function `FindConnectedGroups` selects a triangle, which is not yet in a group and calls the function `FindConnectedTriangles`. Since the returned first list always contains at least one triangle, the length of the list `tr` decreases in every loop call, hence the `FindConnectedGroups` terminates. The question left is how to find all triangles connected with a certain triangle. This has been done in the following way. First we search for all triangles that share at least one node with this triangle. Then we restart the search with all triangles found. For efficiency reasons it is better to perform the search with all triangles we found in one step together. If in one step no further triangles are found then we are ready and return the list of connected triangles and the remaining list. In each step we remove all triangles we found from the list of triangles that are not declared as connected. This increases the speed because the search is faster if there are less elements to compare. More important, this prevent us to search in loops and find some triangles more than once. In general, search in a loop can be the reason for a termination problem but due to shrinking the list of triangles to search for in every step the termination can be guaranteed.

5.3 Results

In Fig. 1 the geometrical structure of the solution set is shown, for the case m_i, $n_i \leq 50$ and $L_x = L_y = 1$.

Below we show all the topological elements of this solution set.

1. Twenty-one groups contain only one triangle (obviously, they have isomorphic dynamical systems):

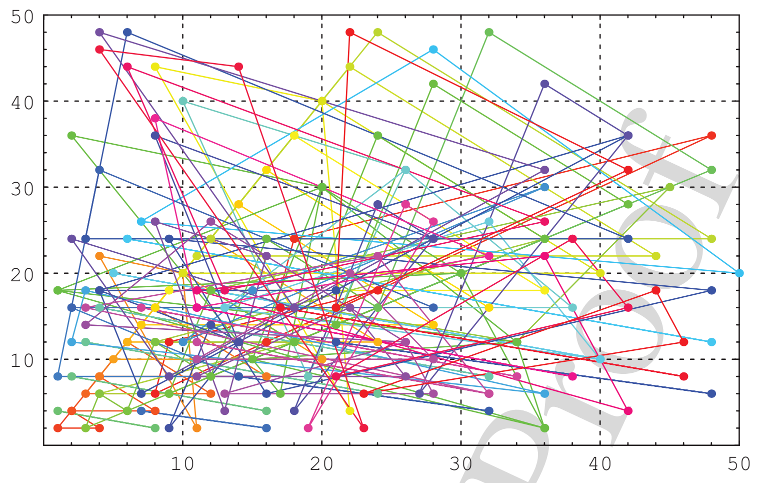


Fig. 1 The geometrical structure of the result in domain $D = 50$

$\{\{3,18\},\{36,6\},\{2,12\}\}$ $\{\{4,46\},\{14,44\},\{23,2\}\}$
$\{\{6,44\},\{36,26\},\{13,18\}\}$ $\{\{6,48\},\{42,24\},\{3,24\}\}$
$\{\{8,26\},\{16,22\},\{13,4\}\}$ $\{\{9,24\},\{48,18\},\{16,6\}\}$
$\{\{14,28\},\{28,14\},\{7,14\}\}$ $\{\{18,36\},\{36,18\},\{9,18\}\}$
$\{\{22,16\},\{26,8\},\{11,8\}\}$ $\{\{22,20\},\{28,10\},\{11,10\}\}$
$\{\{22,44\},\{44,22\},\{11,22\}\}$ $\{\{22,48\},\{42,32\},\{21,16\}\}$
$\{\{24,18\},\{9,12\},\{8,6\}\}$ $\{\{26,28\},\{28,26\},\{19,2\}\}$
$\{\{28,42\},\{42,28\},\{21,14\}\}$ $\{\{28,46\},\{50,20\},\{7,26\}\}$
$\{\{36,22\},\{42,4\},\{11,18\}\}$ $\{\{36,30\},\{15,18\},\{10,12\}\}$
$\{\{38,24\},\{42,16\},\{21,8\}\}$ $\{\{44,18\},\{46,12\},\{23,6\}\}$
$\{\{48,36\},\{18,24\},\{16,12\}\}$

2. Further nine groups also contain one triangle, but in each triangle two points coincide (again, they have isomorphic dynamical systems):

$\{\{8,2\},\{8,2\},\{1,4\}\}$ $\{\{16,2\},\{16,2\},\{7,4\}\}$
$\{\{16,4\},\{16,4\},\{2,8\}\}$ $\{\{24,6\},\{24,6\},\{3,12\}\}$
$\{\{32,8\},\{32,8\},\{4,16\}\}$ $\{\{34,8\},\{34,8\},\{7,16\}\}$
$\{\{46,8\},\{46,8\},\{17,16\}\}$ $\{\{48,6\},\{48,6\},\{21,12\}\}$
$\{\{48,12\},\{48,12\},\{6,24\}\}$

3. There exist two groups with two triangles each (by observation of the geometrical pictures it is easy to determine that both have isomorphic dynamical systems):

$\{\ \{\{2,24\},\{18,16\},\{9,8\}\},\quad \{\{4,48\},\{36,32\},\{18,16\}\}\ \}$
$\{\ \{\{12,26\},\{26,12\},\{3,14\}\},\ \{\{26,12\},\{28,6\},\{13,6\}\}\ \}$

4. Two further groups consist of two triangles each, but the common point is contained twice in one triangle (the dynamical systems are isomorphic, but different from the two groups above):

$$\begin{array}{l}\{\ \{\{24,22\},\{32,6\},\{3,16\}\},\ \ \{\{32,6\},\{32,6\},\{11,12\}\}\ \}\\ \{\ \{\{8,38\},\{32,22\},\{11,16\}\},\ \{\{38,8\},\{38,8\},\{11,16\}\}\ \}\end{array}$$

5. As we can see by inspecting their geometrical structures, further seven groups are not isomorphic to any group found above:

$$\begin{array}{lll}\{\ \{\{6,12\},\{12,6\},\{3,6\}\}, & \{\{12,24\},\{24,12\},\{6,12\}\}, & \\ \ \ \{\{24,48\},\{48,24\},\{12,24\}\} & & \}\end{array}$$

$$\begin{array}{lll}\{\ \{\{2,16\},\{14,8\},\{1,8\}\}, & \{\{4,32\},\{28,16\},\{2,16\}\}, & \\ \ \ \{\{32,4\},\{32,4\},\{14,8\}\} & & \}\end{array}$$

$$\begin{array}{ll}\{\ \{\{2,4\},\{4,2\},\{1,2\}\}, & \{\{4,8\},\{8,4\},\{2,4\}\},\\ \ \ \{\{8,16\},\{16,8\},\{4,8\}\}, & \{\{16,32\},\{32,16\},\{8,16\}\}\ \}\end{array}$$

$$\begin{array}{ll}\{\ \{\{4,22\},\{10,20\},\{11,2\}\}, & \{\{8,44\},\{20,40\},\{22,4\}\},\\ \ \ \{\{10,20\},\{20,10\},\{5,10\}\}, & \{\{20,40\},\{40,20\},\{10,20\}\}\ \}\end{array}$$

$$\begin{array}{ll}\{\ \{\{10,40\},\{26,32\},\{19,8\}\}, & \{\{26,32\},\{38,16\},\{13,16\}\},\\ \ \ \{\{32,26\},\{40,10\},\{13,16\}\}, & \{\{40,10\},\{40,10\},\{5,20\}\}\quad \}\end{array}$$

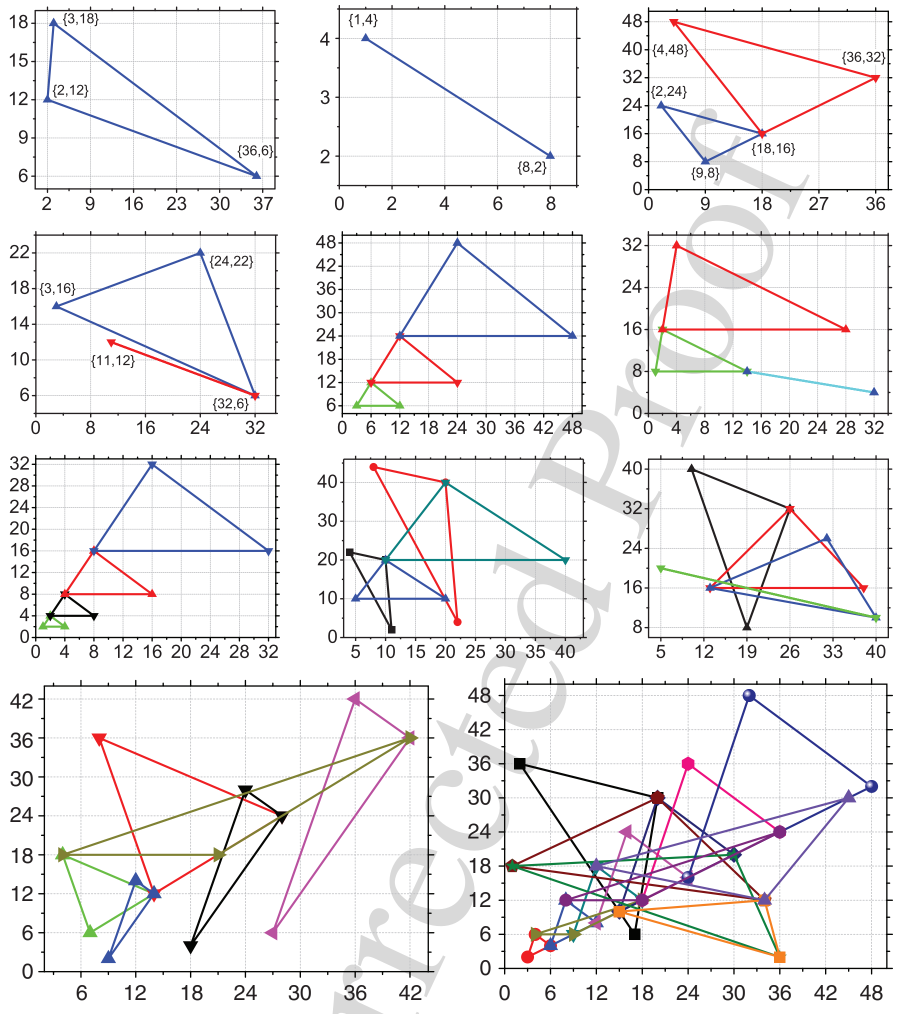
$$\begin{array}{ll}\{\ \{\{4,18\},\{14,12\},\{7,6\}\}, & \{\{8,36\},\{28,24\},\{14,12\}\},\\ \ \ \{\{12,14\},\{14,12\},\{9,2\}\}, & \{\{24,28\},\{28,24\},\{18,4\}\},\\ \ \ \{\{36,42\},\{42,36\},\{27,6\}\}, & \{\{42,36\},\{21,18\},\{4,18\}\}\ \}\end{array}$$

$$\begin{array}{ll}\{\ \{\{2,36\},\{20,30\},\{17,6\}\}, & \{\{4,6\},\{6,4\},\{3,2\}\},\\ \ \ \{\{8,12\},\{12,8\},\{6,4\}\}, & \{\{12,18\},\{18,12\},\{9,6\}\},\\ \ \ \{\{16,24\},\{24,16\},\{12,8\}\}, & \{\{18,12\},\{9,6\},\{4,6\}\},\\ \ \ \{\{20,30\},\{30,20\},\{15,10\}\}, & \{\{20,30\},\{34,12\},\{1,18\}\},\\ \ \ \{\{24,36\},\{36,24\},\{18,12\}\}, & \{\{30,20\},\{36,2\},\{1,18\}\},\\ \ \ \{\{32,48\},\{48,32\},\{24,16\}\}, & \{\{34,12\},\{36,2\},\{15,10\}\},\\ \ \ \{\{36,24\},\{18,12\},\{8,12\}\}, & \{\{45,30\},\{34,12\},\{12,18\}\}\ \}\end{array}$$

Geometrical interpretation of all topological elements is given below. In cases when there exist more then one element with given structure, wave numbers are written at the picture corresponding to the element chosen for presentation.

5.4 Important Remark

To compute all nonisomorphic subgraphs algorithmically is a nontrivial problem. Indeed, all isomorphic graphs presented in previous section are described by similar dynamical systems, only magnitudes of interaction coefficients α_i vary. However, in the general case graph structure thus defined does not present the dynamical system



unambiguously. Consider Fig. 2 below where two objects are isomorphic *as graphs*. However, the first object represents four connected triads with dynamical system

$$(A_1, A_2, A_3), (A_1, A_2, A_5), (A_1, A_3, A_4), (A_2, A_3, A_6), \tag{8}$$

while the second three connected triads with dynamical system

$$(A_1, A_2, A_5), (A_1, A_3, A_4), (A_2, A_3, A_6). \tag{9}$$

This problem has been solved in Kartashova and Mayrhofer (2007) by introducing hypergraphs of a special structure; the standard graph isomorphism algorithm used by Mathematica has been modified in order to suit hypergraphs.

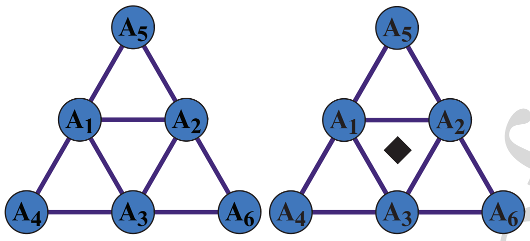


Fig. 2 Example of isomorphic graphs and nonisomorphic dynamical systems. The left graph corresponds to the dynamical system (8) and the graph on the right to the dynamical system (9). To discern between these two cases we set a placeholder inside the triangle not representing a resonance

6 A Web Interface to the Software

The previous sections have presented implementations of various symbolic computation methods for the analysis of nonlinear wave resonances. These implementations are written in the language of the computer algebra system Mathematica, which provides an appealing graphical user interface (GUI) for executing computations and presenting the results. For instance, the pictures shown in Sect. 4.3 were produced by converting the computed hypergraphs to Mathematica plot structures that can be displayed by the GUI of the system.

However, to run these methods the user needs an installation of Mathematica on the local computer with the previously described methods installed in a local directory. These requirements make access to the software difficult and hamper its wide-spread usage. To overcome this problem, we have implemented a Web interface such that the software can be executed from any computer connected to the Internet via a Web browser without the need for a local installation of mathematical software.

This implementation follows a general trend in computer science, which turns away from stand alone software (that is installed on local computers and can be executed only on these computers via a graphical user interface) and proceeds towards *service-oriented software* (Gold et al. 2004) (that is installed on remove server computers and wraps each method into a service that can be invoked over the Internet via standardized Web interfaces). Various projects in computer mathematics have pursued middleware for *mathematical web services*, see for instance MathBroker (2007), MONET (2004), Baraka and Schreiner (2006). On the long term, it is thus envisioned that mathematical methods generally become remote services that can be invoked by humans (or other software) without requiring local software installations.

However, even without sophisticated middleware it is nowadays relatively simple to provide (for restricted application scenarios) web interfaces to mathematical software by generally available technologies. The web interface presented in the following sections is deliberately kept as simple as possible and makes only use of such technologies; thus it should be easy to take this solution as a blueprint for other mathematical software with similar features. In particular, the web interface is quite independent of Mathematica as the system underlying the implementation of the mathematical methods; the same strategy can be applied to other mathematical software systems such as Maple, MATLAB, etc.

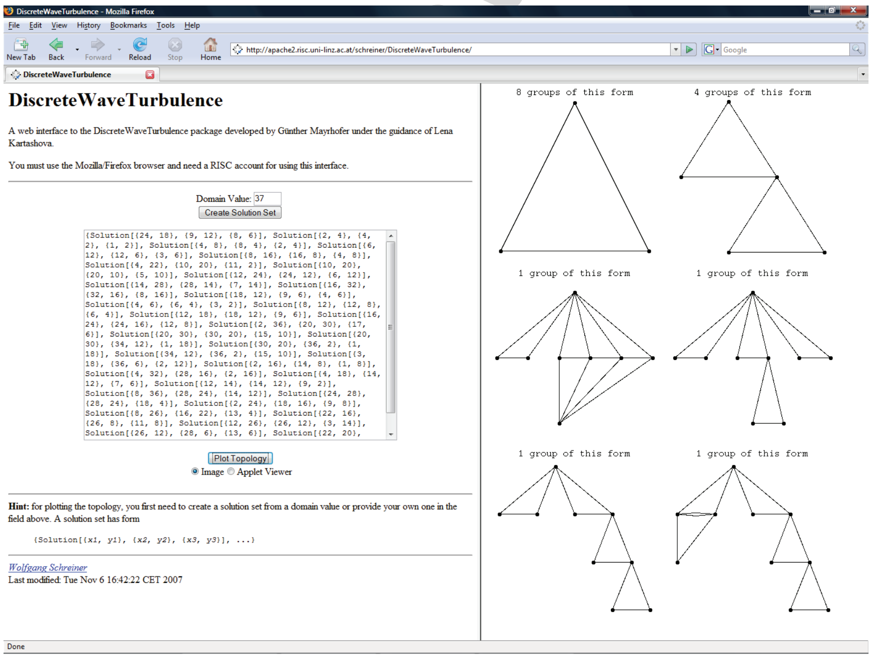
6.1 The Interface

Figure 3 shows the web interface to some of the methods presented in the previous sections. Its functionality is as follows:

Create Solution Set: The user may enter a parameter D in the first (small) text field and then press the button "Create Solution Set." This invokes the method `CreateSolutionSet`, which computes the set of all solutions whose values are smaller than or equal to D. This set is written into the second (large) text field in the form

$$\{\texttt{Solution[}x_1\texttt{,}y_1\texttt{,}z_1\texttt{]},\texttt{...},\texttt{Solution[}x_n\texttt{,}y_n\texttt{,}z_n\texttt{]}\}.$$

Plot Topology: The user may enter into the second (large) text field a specific solution set (or, as show above, compute one), and then press the button "Plot Topology." This first invokes the method `Topology`, which computes the topological structure of the solution set as a list of hypergraphs and then calls the



DiscreteWaveTurbulence

A web interface to the DiscreteWaveTurbulence package developed by Günther Mayrhofer under the guidance of Lena Kartashova.

You must use the Mozilla/Firefox browser and need a RISC account for using this interface.

```
{Solution[{24, 18}, {9, 12}, {8, 6}], Solution[{2, 4}, {4,
2}, {1, 2}], Solution[{4, 8}, {8, 4}, {2, 4}], Solution[{6,
12}, {12, 6}, {3, 6}], Solution[{8, 16}, {16, 8}, {4, 8}],
Solution[{4, 22}, {10, 20}, {11, 2}], Solution[{10, 20},
{20, 10}, {5, 10}], Solution[{12, 24}, {24, 12}, {6, 12}],
Solution[{14, 28}, {28, 14}, {7, 14}], Solution[{16, 32},
{32, 16}, {8, 16}], Solution[{18, 12}, {9, 6}, {4, 6}],
Solution[{4, 6}, {6, 4}, {3, 2}], Solution[{8, 12}, {12, 8},
{6, 4}], Solution[{12, 18}, {18, 12}, {9, 6}], Solution[{16,
24}, {24, 16}, {12, 8}], Solution[{2, 36}, {20, 30}, {17,
6}], Solution[{20, 30}, {30, 20}, {15, 10}], Solution[{20,
30}, {34, 12}, {1, 18}], Solution[{30, 20}, {36, 2}, {1,
18}], Solution[{34, 12}, {36, 2}, {15, 10}], Solution[{3,
18}, {36, 6}, {2, 12}], Solution[{2, 16}, {14, 8}, {1, 8}],
Solution[{4, 32}, {28, 16}, {2, 16}], Solution[{4, 18}, {14,
12}, {7, 6}], Solution[{12, 14}, {14, 12}, {9, 2}],
Solution[{8, 36}, {28, 24}, {14, 12}], Solution[{24, 28},
{28, 24}, {18, 4}], Solution[{2, 24}, {18, 16}, {9, 8}],
Solution[{8, 26}, {16, 22}, {13, 4}], Solution[{22, 16},
{26, 8}, {11, 8}], Solution[{12, 26}, {26, 12}, {3, 14}],
Solution[{26, 12}, {28, 6}, {13, 6}], Solution[{22, 20},
```

Hint: for plotting the topology, you first need to create a solution set from a domain value or provide your own one in the field above. A solution set has form

```
{Solution[{x1, y1}, {x2, y2}, {x3, y3}], ...}
```

Wolfgang Schreiner
Last modified: Tue Nov 6 16:42:22 CET 2007

Fig. 3 Web interface to the implementation

method `PlotTopology`, which computes a plot of each hypergraph. The results are displayed in the right frame of the browser window.

The web interface is available at the URL

`http://www.risc.uni-linz.ac.at/projects/alisa`
(Button "Discrete Wave Turbulence")

To run the computations, an account and a password are needed.

6.2 The Implementation

The web interface is implemented in PHP, a scripting language for producing dynamic web pages (The PHP Group 2007). PHP scripts can be embedded into conventional HTML pages within tags of form `<php?...?>`; when a Web browser requests such a page, the Web server executes the scripts with the help of an embedded PHP engine, replaces the tags by the generated output, and returns the resulting HTML page to the browser. With the use of PHP, thus programs can be implemented that run on a web server and deliver their results to a client computer which displays them in a web browser. The web interface to the discrete wave turbulence package is implemented in PHP as sketched in Fig. 4 and described below (the parenthesized numbers in the text refer to the corresponding numbers in the figure).

Create Solution Set: The browser frame `input` on the left side contains essentially the following HTML input form:

```
 <form target="textarea"
     action="https://apache2.../CreateSolutionSet.php"
     method="post">
   <input name="domain" size="3">
   <input type="submit" value="Create Solution Set">
</form>
```

This form consists of an input field `domain` to receive a domain value and a button to trigger the creation of the solution set. When the button is pressed, (1) a request is sent to the web server which carries the value of `domain`; this request asks the server to deliver the PHP-enhanced web page `CreateSolutionSet.php` into the target frame `textarea`, which is displayed internally to `input`.

The file `CreateSolutionSet.php` has essentially the content

```
<?php
  $math="/.../math";
  $cwd="/.../DiscreteWaveTurbulence";
  $domain = $_POST['domain'];
  $mcmd =
    "SetDirectory[\"" . $cwd . "\"]; " .
    "Needs[\"DiscreteWaveTurbulence`SolutionSet`\"]; " .
    "sol=DiscreteWaveTurbulence`SolutionSet`CreateSolutionSet[".
```

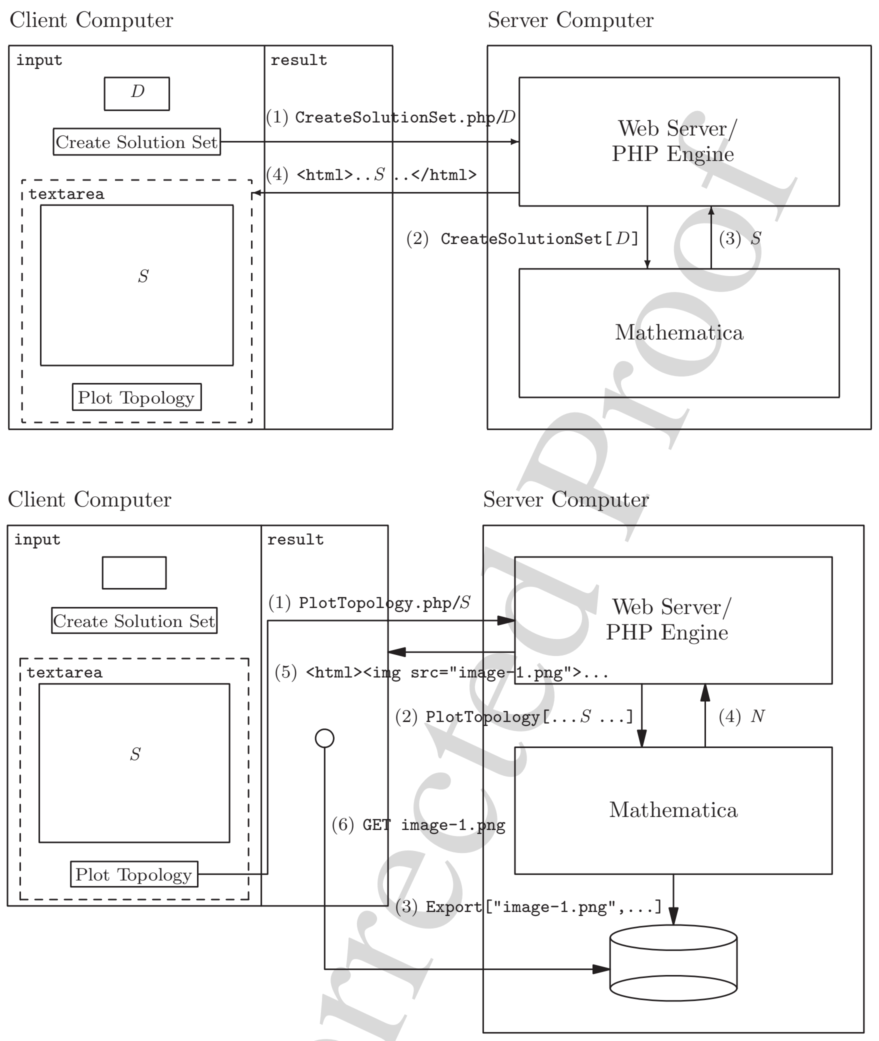


Fig. 4 Implementation of the web interface

```
       $domain . "]; ";
  $command="$math -noprompt -run '" . $mcmd .
     "Print[StandardForm[sol]]; Quit[];'";
  $result = shell_exec("$command");
  echo
    ...
    "<textarea  name=\"sol\" cols=\"60\" rows=\"20\">" .
    htmlspecialchars($result) .
    "</textarea>" .
   ...;
 ?>
```

After setting the paths `$math` of the Mathematica binary and `$cwd` of the directory where the `DiscreteWaveTurbulence` package is installed, the script sets the local variable `$domain` to the value of the input field `domain`. Then the Mathematica command `$mcmd` is constructed in order to load the file `SolutionSet.m` and execute the command `CreateSolutionSet` to compute the solution set. Now the system command `$command` is constructed to (2) invoke Mathematica, which calls the previously constructed command, and (3) prints its result to the standard output stream, which is captured in the variable `$result`. From this, the script contstructs the HTML code of the result document, which is (4) delivered to the Web browser.

Plot Topology: The browser frame `textarea` contains essentially the following HTML input form:

```
<form target="result"
    action="https://apache2..../PlotTopology.php"
    method="post">
  <textarea name="sol" cols="60" rows="20">...</textarea>
  <input type="submit" value="Plot Topology">
  </center>
</form>
```

This form consists of the textarea field `sol` to receive the solution set and a button to trigger the plotting of the topology of this set. When the button is pressed, (1) a request is sent to the web server which carries the value of `sol`; this request asks the server to deliver the PHP-enhanced web page `PlotTopology.php` into the target frame `result` on the right side of the browser.

The file `CreateSolutionSet.php` has essentially the content

```
<?php
  $math="/.../math";
  $basedir ="/.../DiscreteWaveTurbulence";
  $baseurl ="http://apache2/.../DiscreteWaveTurbulence";
  $sol = $_POST['sol'];
  ... // create under $basedir a unique subdirectory $dir
  $mcmd =
    "SetDirectory[\"$basedir/$dir\"]; " .
    "Needs[\"DiscreteWaveTurbulence`Topology`\"]; " .
    "Needs[\"DiscreteWaveTurbulence`SolutionSet`\"]; " .
    "top=DiscreteWaveTurbulence`Topology`Topology[$sol]; " .
    "plots=DiscreteWaveTurbulence`Topology`PlotTopology1[top];";
  $command="/usr/bin/Xvnc :20 & export DISPLAY=:20; " .
    "export MATHEMATICA_USERBASE=$basedir/.Mathematica; " .
    "$math -run '" . $mcmd .
    "Print[ExportList[plots,\"$image\"]]; Quit[];'";
  $result = shell_exec("$command | tail -n 1");
  for ($i=0;$i<$result;$i++)
    echo "<img src=\"$baseurl/$dir/image-$i.png\"/>";
?>
```

For holding the images to be generated later, the script creates a unique directory `$basedir/$dir`, which is served by the web server under the url

`$baseurl/$dir`. The script extracts the solution set `$sol` from the request and sets up the Mathematica command to compute its topological structure and generate the plots from which ultimately the image files will be produced.

For this purpose, however, Mathematica needs an X11 display server running; since a Web server has no access to an X11 server, we start the virtual X11 server Xvnc (RealVNC Remote Control Software 2007) as a replacement and set the environment variable `DISPLAY` to the display number on which the number listens; Mathematica will subsequently send X11 requests to that display, which will be handled by the virtual server. Likewise, Mathematica needs access to a `.Mathematica` configuration directory; the script sets the environment variable `MATHEMATICA_USERBASE` correspondingly.

With these provisions, we can (2) invoke first the command to compute the plots and then the (self-defined) command `ExportList` to generate for every plot an image in the previously created directory. For this purpose the command uses (3) the Mathematica command `EXPORT[`*`file`*`,`*`plot`*`,"PNG"]`, which converts *plot* to an image in PNG format and writes the image to *file*. `ExportList` returns the number of images generated, which is (4) written to the standard output stream which in turn is captured in the variable `$result`. From this information, the script generates an HTML document, which contains a sequence of `img` elements referencing these images. After this document has been (5) returned to the client browser, the browser (6) requests the referenced images with `GET` messages from the web server.

6.3 Extensions

As an alternative to the display of static images, the Web interface also provides an option “Applet Viewer” with somewhat more flexibility. If this option is selected, Mathematica is instructed to save all generated plots as files in the standard representation. The generated HTML document then embeds (rather than `img` elements) a sequence of `applet` elements that load instances of the “JavaView” applet (The JavaView Project 2007). These applets run in the Java Virtual Machine of the Web browser on the client computer, load the plot files from the web, and visualize them in the browser. Rather than just displaying static images, the viewer allows to perform certain manipulations and transformations of the plots such as scaling, rotating, etc. While this additional flexibility is not of particular importance for the presented methods, they may in the future become useful for others.

To limit access to the software, respectively, to the computing power of the server computer, it may be protected by authentication mechanisms. For example, on the Apache Web server, it suffices to provide in the installation directory of the software a file `.htaccess` with content

```
<Files "*.php">
  SSLRequireSSL
  AuthName "your account"
```

```
    AuthType Basic
    Require valid-user
  </Files>
```

With this configuration, the user is asked for the data of a valid account on the computer running the Web server; other authentication mechanisms based, for example, on password files may be provided in a similar fashion.

7 Discussion

Summing up all the results obtained, we would like to make some concluding remarks.

- In general, coefficients α_i can be computed symbolically by hand and only numerically by Mathematica (see Sect. 3.3); at present we are not aware of the possibility to overcome this problem.
- For the known case of spherical barotropic vorticity equation, values of coefficients α_i coincide with known form of the literature for all triads except three. These three triads, though satisfying resonant conditions, are known to be special from the physical point of view in the following sense (see Kartashova and L'vov (2007) for details). Though resonance conditions are fulfilled for the waves of these triads, they, so to say, do not have a place in the physical space to interact and their influence (if any) on the dynamics of the wave system has to be studied separately from all other waves. Our results might indicate that also the coefficients α_i of these triads have to be defined in some other way compared to other resonant triads. For instance, another way of space-averaging has to be chosen.
- The results of Sect. 3.4.2 show that analytical formulae given in Kartashova and Reznik (1992) for α_j *are not correct.*
- The results of Sect. 4.3 show a crucial dependence of the number of solutions on the form of the boundary conditions. In particular, some boundary conditions (for example, $(L_x, L_y) = (11, 29)$) yield *no solutions*, which is of most importance for physical applications. From the mathematical point of view, an interesting result has been observed: in all our computations (i.e., for $m, n \leq 300$) indexes corresponding to nonempty classes turned out to be *odd.* It would be interesting to prove this fact analytically because if it is true, we can reduce the computational time.
- The algorithm presented in Sect. 4 has been implemented before numerically in Visual Basic, and our purpose here was to show that it works fast enough also in Mathematica. The algorithms presented in Sects. 3 and 5 *have never been implemented before*, the whole work is usually done by hand and some mistakes as in Kartashova and Reznik (1992) are almost unavoidable: it takes sometimes a few weeks of skillful researchers to compute interaction coefficients of dynamical systems for one specific wave system.

- All the algorithms presented above can easily be modified for the case of a four-term mesoscopic system. The only problem left is a procedure to establish all nonisomorphic topological elements for a quadruple graphs, similar to the procedure given in Kartashova and Mayrhofer (2007) for a triangle graphs. The structure of quadruple graphs is much more complicated while some mechanisms of energy transfer in the spectral space do exist (Kartashova 2007) that are absent in three-term mesoscopic systems. A complete classification of quadruple graphs is still an open question but in a given spectral domain it can be done directly (a very time consuming operation).
- We have developed a Web interface for the presented methods, which turns the implementations from only locally available software to Web-based services that can be accessed from any computer in the Internet that is equipped with a Web browser. The presented implementation strategy is simple and is based on generally available technologies; it can be applied as a blueprint for a large variety of mathematical software. In particular, the results are not bound to the current Mathematica implementation but can be adapted to any other computer algebra system (e.g., Maple) or numerical software system (e.g., MATLAB) of similar expressiveness.
- At present, an explicit form of eigen-modes (5) and (6) is used as one of the input parameters for our program package. Theoretically, at least for some classes of linear partial differential operators and boundary conditions, computing eigen-modes can also be performed symbolically based on the results in Rosenkranz (2005). If this were done, not an eigen-mode but boundary conditions would play role of input parameter.

Acknowledgements Authors acknowledge the support of the Austrian Science Foundation (FWF) under projects SFB F013/F1301 "Numerical and symbolical scientific computing," P20164-N18 "Discrete resonances in nonlinear wave systems," and P17643-NO4 "MathBroker II: Brokering Distributed Mathematical Services."

References

R. Baraka, W. Schreiner. "Semantic Querying of Mathematical Web Service Descriptions." Third International Workshop on Web Services and Formal Methods (WS-FM 2006), Vienna, Austria, September 8–9, 2006. M. Bravetti, et al. (eds.), Lecture Notes in Computer Science, vol. 4184 (Springer, Berlin Heidelberg New York), pp. 73-87

G.P. Berman, F.M. Israilev. "The Fermi-Pasta-Ulam problem: Fifty years of progress." *Chaos* **15** (1), 015104–015104-18 (2005)

M. Cheney. *Tesla Man Out of Time* (Dorset Press, New York, 1989)

N. Gold, A. Mohan, C. Knight, M. Munro. "Understanding service-oriented software." IEEE Software, **21**(2) 71–77 (2004).

E.A. Kartashova. "Wave resonances in systems with discrete spectra." In: V.E. Zakharov (ed.) *Nonlinear Waves and Weak Turbulence*, Advances in the Mathematical Sciences (AMS, 1998), pp. 95–129

E. Kartashova. "A model of laminated turbulence." *JETP Lett.* **83** (7), 341–345 (2006a)

E. Kartashova. "Fast computation algorithm for discrete resonances among gravity waves." *Low Temp. Phys.* **145**(1–4), 286–295 (2006b)

E. Kartashova. "Exact and quasi-resonances in discrete water-wave turbulence." *Phys. Rev. Lett.* **98**(21) 214502 (2007)

E. Kartashova, A. Kartashov. "Laminated wave turbulence: Generic algorithms I." *Int. J. Mod. Phys. C* **17**(11), 1579–1596 (2006)

E. Kartashova, A. Kartashov. "Laminated wave turbulence: Generic algorithms II." *Comm. Comp. Phys.* **2**(4), 783–794 (2007a)

E. Kartashova, A. Kartashov. "Laminated wave turbulence: Generic algorithms III." *Phys. A: Stat. Mech. Appl.* **380**, 66–74 (2007b)

E. Kartashova, V.S. L'vov. "A model of intra-seasonal oscillations in the Earth atmosphere." *Phys. Rev. Lett.* **98** (19) 198501 (2007) (featured in Nature Physics **3**(6) 368, 2007)

E. Kartashova, G. Mayrhofer. "Cluster formation in mesoscopic systems." *Phys. A: Stat. Mech. Appl.* **385**, 527–542 (2007)

E.A. Kartashova, G.M. Reznik. "Interactions between Rossby waves in bounded regions." *Oceanology* **31**, 385–389 (1992)

MathBroker II: Brokering Distributed Mathematical Services Reseach Institute for Symbolic Computation (RISC), 2007. http://www.risc.uni-linz.ac.at/projects/mathbroker2

MONET – Mathematics on the Web. The MONET Consortium, April 2004. http://monet.nag.co.uk

A.N. Nayfeh. *Introduction to Perturbation Techniques.* (Wiley, New York, 1981)

J. Pedlosky. *Geophysical Fluid Dynamics* (Springer, Berlin Heidelberg New York, 1987)

A.N. Pushkarev, V.E. Zakharov. "Turbulence of capillary waves – theory and numerical simulations." *Physica D* **135**, 98–116 (2000)

RealVNC Remote Control Software. "VNC Free Edition 4.1." http://www.realvnc.com/products/free/4.1, 2007

M. Rosenkranz. "A new symbolic method for solving linear two-point boundary value problems on the level of operators." *J. Symb. Comp.* **39**, 171–199 (2005)

M. Tanaka. "On the role of resonant interactions in the short-term evolution of deep-water ocean spectra." *J. Phys. Oceanogr.* **37**, 1022–1036 (2007)

The JavaView Project. "JavaView – Interactive 3D Geometry and Visualization." http://www.javaview.de, 2007

The PHP Group. "PHP: Hypertext Preprocessor." http://www.php.net, 2007

V. Zakharov, F. Dias, A. Pushkarev. "One-dimensional wave turbulence." *Phys. Rep.* **398**, 1–65 (2004)

V.E. Zakharov, N.N. Filonenko. "Weak turbulence of capillary waves." *J. Appl. Mech. Tech. Phys.* **4**, 500–515 (1967)

V.E. Zakharov, A.O. Korotkevich, A.N. Pushkarev, A.I. Dyachenko. "Mesoscopic wave turbulence." *JETP Lett.*, **82**(8), 491 (2005)

V.E. Zakharov, V.S. L'vov, G. Falkovich. *Kolmogorov Spectra of Turbulence.* Series in Nonlinear Dynamics (Springer, Berlin Heidelberg New York, 1992)