Shallow water model documentation

Milan Kloewer **version of March 5, 2017**

This documentation aims to summarize a methodology how the shallow water equations can be discretized and solved numerically with finite differences. The corresponding numerical model can be found at `www.github.com/milankl/swm`.

1 Physical model

1.1 General model

The shallow water equations of interest are (see Gill?)

$$\partial_t u + u\partial_x u + v\partial_y u - fv = -g\partial_x \eta + F_x + M_x \tag{1a}$$
$$\partial_t v + u\partial_x v + v\partial_y v + fu = -g\partial_y \eta + F_y + M_y \tag{1b}$$
$$\partial_t \eta + \partial_x(uh) + \partial_y(vh) = 0 \tag{1c}$$

with

$\mathbf{u} = (u,v) = (u(x,y,t), v(x,y,t))$	horizontal velocity vector
$\eta = \eta(x,y,t)$	surface displacement
$h = h(x,y,t) = \eta + H$	layer thickness
$H = H(x,y)$	undisturbed layer thickness
$f = f(y)$	coriolis parameter
$g = \text{const}$	gravitational acceleration
$\mathbf{F} = (F_x, F_y) = (F_x(x,y,t), F_y(x,y,t))$	forcing vector
$\mathbf{M} = (M_x, M_y) = (M_x(u,v,h), M_y(u,v,h))$	lateral mixing of momentum, or friction term

and differential operators

$$\partial_x = \frac{\partial}{\partial x}, \partial_y = \frac{\partial}{\partial y}, \partial_t = \frac{\partial}{\partial t}$$

on the domain $\mathcal{D} = (0, L_x) \times (0, L_y)$ of width (or east-west extent) L_x and north-south extent L_y and with cartesian coordinates x, y and time t. The initial conditions are

$$u(t=0) = u_0(x,y) \tag{2a}$$
$$v(t=0) = v_0(x,y) \tag{2b}$$
$$h(t=0) = h_0(x,y) \tag{2c}$$

The kinematic boundary condition states that there is no flow through the boundary, i.e.

$$u(x=0) = u(x=L_x) = 0 \tag{3a}$$
$$v(y=0) = v(y=L_y) = 0, \tag{3b}$$

additionally we require no gradient of η across the boundary, hence

$$\partial_x\eta(x=0) = \partial_x\eta(x=L_x) = 0 \tag{4a}$$
$$\partial_y\eta(y=0) = \partial_y\eta(y=L_y) = 0, \tag{4b}$$

and the tangential velocity at the boundary should vanish, which is usually referred to as no-slip boundary conditions

$$v(x=0) = v(x=L_x) = 0 \tag{5a}$$
$$u(y=0) = u(y=L_y) = 0. \tag{5b}$$

However, sometimes these equations are replaced by

$$\partial_x v(x=0) = \partial_x v(x=L_x) = 0 \tag{6a}$$
$$\partial_y u(y=0) = \partial_y u(y=L_y) = 0, \tag{6b}$$

which corresponds physically to a friction-less boundary and are therefore called *free-slip* boundary conditions.

The prognostic variables so far are u, v, η, however, as η only appears in gradients, also h can be used as prognostic variable. We can separate from the advective terms in equations 1a,b the spatial gradient of kinetic energy. In combination with the pressure gradient term we introduce the Bernoulli potential p as

$$p = \frac{1}{2}(u^2+v^2) + gh \tag{7}$$

Furthermore, with introducing the relative vorticity $\zeta = \partial_x v - \partial_y u$ the potential vorticity can be defined as

$$q = \frac{f+\zeta}{h} \tag{8}$$

and the equations 1a,b,c then become

$$\partial_t u = qhv - \partial_x p + F_x + M_x \tag{9a}$$
$$\partial_t v = -qhu - \partial_y p + F_y + M_y \tag{9b}$$
$$\partial_t h = -\partial_x(uh) - \partial_y(vh) \tag{9c}$$

which are the equations solved by the numerical model as described in the next section.

1.2 Double gyre set up

In order to simulate mid-latitudinal dynamics we choose the physical parameters of the previous section as

$$\begin{aligned}
g &= 10 \text{ m/s} \\
H &= 500 \text{ m} \\
L_x = L_y &= 3840 \text{ km} \\
F_y &= 0 \\
F_x &= \frac{F_0}{\rho_0 H}\left[\cos\left(2\pi\left(\frac{y}{L_y}-\frac{1}{2}\right)\right) + 2\sin\left(2\pi\left(\frac{y}{L_y}-\frac{1}{2}\right)\right)\right] \\
F_0 &= 0.12 \text{ Pa} \\
\rho_0 &= 1000 \text{ kgm}^{-3}
\end{aligned}$$

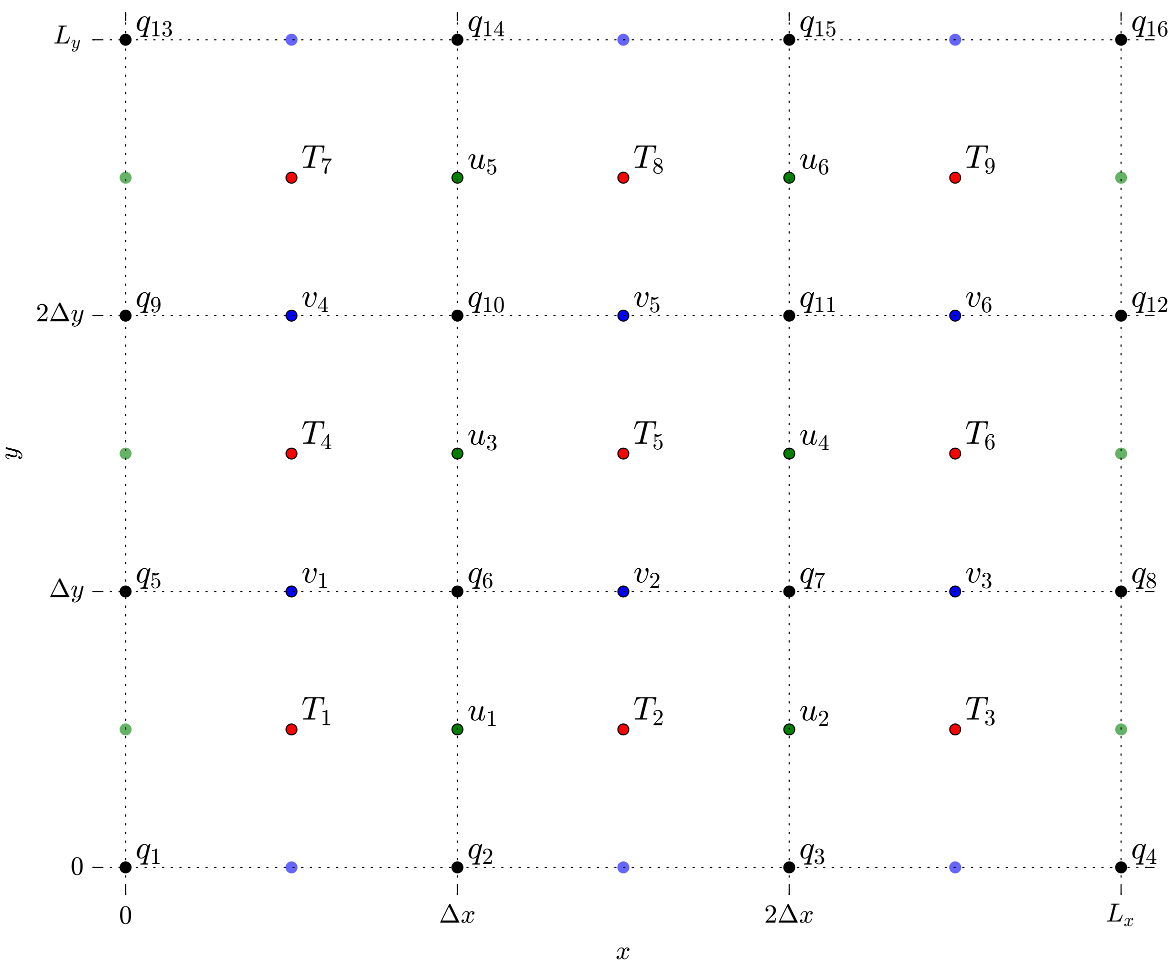

Figure 1: Grid cell numbering following the ideas of an Arakawa C-grid. The values of u, v vanish at the faint grid nodes on the boundary (kinematic boundary condition, see equation 5 and 6) and therefore these grid nodes are not explicitly resolved.

and start from rest, so that the initial conditions become

$$u_0 = v_0 = 0, \quad h_0 = H \tag{10}$$

2 Spatial discretization

2.1 Grid

The domain $\mathcal{D}$ is divided into $n_x \times n_y$ grid cells, evenly spaced, so that each grid cell has the side length $\Delta x = \frac{L_x}{n_x}$ in x-direction and $\Delta y = \frac{L_y}{n_y}$ in y-direction. The total amount of grid cells is $n_x n_y$. Let the variable h_i sit in the middle of the i-th grid cell at position (x_i, y_i), i.e. $h_i = h(x_i, y_i)$. We use only one index i and with row-first indexing (see

Fig. 1), so we have for $i \in \{1, ..., n_x n_y\}$

$$x_i = \frac{\Delta x}{2}(1 + (i-1) \bmod n_x) \tag{11a}$$

$$y_i = \frac{\Delta y}{2}\left(1 + 2\,\mathrm{floor}\left(\frac{i-1}{n_x}\right)\right) \tag{11b}$$

with $a \bmod b$ being the modulo operator and floor(a) is the largest integer not greater than a. Following the ideas of the Arakawa C-grid, the discretization of the variables u, v and q is staggered. In general, we might use an independent indexing for u, v and q and therefore introduce j, k, l

$$u_j = u(x_i + \frac{\Delta x}{2}, y_i) \tag{12a}$$

$$v_k = u(x_i, y_i + \frac{\Delta y}{2}) \tag{12b}$$

$$q_l = q(x_i + \frac{\Delta x}{2}, y_i + \frac{\Delta y}{2}). \tag{12c}$$

Hence, we distinguish between 4 different grids: (i) the T-grid, for h or tracers, (ii) the u-grid, (iii) the v-grid, (iv) the q-grid. Not for all grid cells it is necessary to evaluate u or v, as they might vanish due to the kinematic boundary condition. The grids therefore carry a different amount of grid points. Let N_T, N_u, N_v, N_q be the total number of grid points on the respective grids then

$$\begin{aligned} N_T &= n_x n_y, & N_u &= (n_x - 1)n_y \\ N_q &= (n_x + 1)(n_y + 1), & N_v &= n_x(n_y - 1) \end{aligned} \tag{13}$$

The n_x-th column of u-points vanish, as does the n_y-th row of v-points. However, there is no boundary condition for q, which makes it necessary to evaluate the q-grid for all points within the domain $\mathcal{D}$.

Choosing one index for the grid points leads to the advantage that every scalar variable can be represented as a vector. Numbering the grids row-first as can be seen in Fig. 1, leads to the following vector-representation $\mathtt{u}, \mathtt{v}, \mathtt{h}, \mathtt{q}$ of u, v, h and q

$$\mathtt{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N_u} \end{pmatrix}, \quad \mathtt{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N_v} \end{pmatrix}, \quad \mathtt{h} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_{N_T} \end{pmatrix}, \quad \mathtt{q} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_{N_q} \end{pmatrix}. \tag{14}$$

In the special case of $n_x = n_y$ the vectors $\mathtt{u}, \mathtt{v}$ are of same size, but in general $\mathtt{u}, \mathtt{v}, \mathtt{h}, \mathtt{q}$ all differ in their sizes. Unfortunately, the vector-representation of the variables leads to the necessity to distinguish between grid nodes that are in vicinity of the boundaries and those in the middle of the domain. The distance to the boundary is considered to be the vicinity in case the discrete stencil to evaluate an operation is affected by the boundary. On the u-grid we have for $j \in \{1, ..., N_u\}$

$$x_j = \Delta x(1 + (j-1) \bmod (n_x - 1)) \tag{15a}$$

$$y_j = \frac{\Delta y}{2}\left(1 + 2\,\mathrm{floor}\left(\frac{j-1}{n_x - 1}\right)\right) \tag{15b}$$

On the v-grid this is for $k \in \{1, ..., N_v\}$

$$x_k = \frac{\Delta x}{2}(1 + (k-1) \bmod n_x) \tag{16a}$$

$$y_k = \Delta y \left(1 + \text{floor}\left(\frac{k-1}{n_x}\right)\right) \tag{16b}$$

and on the q-grid this becomes with $l \in \{1, ..., N_q\}$

$$x_l = \Delta x(l \bmod (n_x + 1)) \tag{17a}$$

$$y_l = \Delta y \, \text{floor}\left(\frac{l}{n_x + 1}\right). \tag{17b}$$

2.2 Gradients

Representing the model variables in vector-form, as discussed above, enables us to think of a gradient ∂ as a linear map between two vector spaces V_1, V_2. Hence, any gradient ∂ in that sense can be written as a matrix $\mathbf{G}$ which is multiplied with a vector z representing one of the model's variables:

$$\partial : V_1 \to V_2, \mathsf{z} \to \mathbf{G}\mathsf{z} \tag{18}$$

Having 4 different grids, we have to deal with four different vector spaces V_u, V_v, V_T, V_q. Let K be the algebraic field of the floating-point numbers with a given precision (e.g. 64bit), then $V_u = K^{N_u}, V_v = K^{N_v}, V_T = K^{N_T}$ and $V_q = K^{N_q}$. For simplicity we might consider $K = \mathbb{R}$, i.e. the real numbers. In the following we will investigate the entries of $\mathbf{G}$ such that they describe centred finite differences on the four different grids.

$$\partial_x h|_{\mathbf{x}=\mathbf{x}_j} \approx \frac{h_{j+r+1} - h_{j+r}}{\Delta x}, \quad \text{with } r = \text{floor}(\frac{j-1}{n_x - 1}), \quad \forall\, j \in \{1, ..., N_u\} \tag{19}$$

$$\partial_y h|_{\mathbf{x}=\mathbf{x}_k} \approx \frac{h_{k+n_x} - h_k}{\Delta x}, \quad \forall\, k \in \{1, ..., N_v\} \tag{20}$$

$$\partial_x u|_{\mathbf{x}=\mathbf{x_i}} \approx \begin{cases} \frac{u_i - 0}{\Delta x} & \text{if } (i \bmod n_x) = 1 \\ \frac{0 - u_{i-1}}{\Delta x} & \text{if } (i \bmod n_x) = 0 \\ \frac{u_i - u_{i-1}}{\Delta x} & \text{else.} \end{cases}, \quad \forall\, i \in \{1, ..., N_T\} \tag{21}$$

$$\partial_y v|_{\mathbf{x}=\mathbf{x_i}} \approx \begin{cases} \frac{v_i - 0}{\Delta x} & \text{if } i \leq n_x \\ \frac{0 - v_{i-n_x}}{\Delta x} & \text{if } i > N_y \\ \frac{v_i - v_{i-n_x}}{\Delta x} & \text{else.} \end{cases}, \quad \forall\, i \in \{1, ..., N_T\} \tag{22}$$

In the following we will use a notation where the subscript denotes the direction of the derivative, hence x or y, and the superscript u, v, T, q denotes the vector space, where ∂ is mapping from. For the x-derivative on the T-grid

$$\partial_x^T : V_T \to V_2, \mathsf{z} \to \mathbf{G}_x^T \mathsf{z} \tag{23}$$

(using centred finite differences we will later identify that here $V_2 = V_u$.)

All gradients are now approximated by centred finite differences and

$$\partial_t u = \mathbf{I}_q^u \mathtt{q} * \mathbf{I}_v^u (v * \mathbf{I}_T^v \mathtt{h}) - \mathbf{G}_x^T \mathtt{p} + \mathtt{F}_x + \nu_B \mathbf{L}_u^2 \mathtt{u} \tag{24a}$$

$$\partial_t v = -\mathbf{I}_q^v \mathtt{q} * \mathbf{I}_u^v (u * \mathbf{I}_T^u \mathtt{h}) - \mathbf{G}_y^T \mathtt{p} + \nu_B \mathbf{L}_v^2 \mathtt{v} \tag{24b}$$

$$\partial_t h = -\mathbf{G}_x^u (\mathtt{u} * \mathbf{I}_T^u \mathtt{h}) - \mathbf{G}_y^v (\mathtt{v} * \mathbf{I}_T^v \mathtt{h}) \tag{24c}$$

with

$$\mathtt{p} = \frac{1}{2}\left(\mathbf{I}_u^T(\mathtt{u}^2) + \mathbf{I}_v^T(\mathtt{v}^2)\right) + g\mathtt{h} \tag{25a}$$

$$\mathtt{q} = \frac{\mathtt{f}_q + \mathbf{G}_x^v \mathtt{v} - \mathbf{G}_y^u \mathtt{u}}{\mathbf{I}_T^q \mathtt{h}} \tag{25b}$$

3 Time discretization

3.1 Runge-Kutta 4th order

The discrete shallow water model is integrated forward in time with the 4th order Runge-Kutta scheme (RK4). Summarizing the right-hand side of equations 24 with rhs$(\mathtt{u},\mathtt{v},\mathtt{h}) = (\mathtt{du},\mathtt{dv},\mathtt{dh})$ the model equations reduce to

$$\partial_t \begin{pmatrix} \mathtt{u} \\ \mathtt{v} \\ \mathtt{h} \end{pmatrix} = \begin{pmatrix} \mathtt{du} \\ \mathtt{dv} \\ \mathtt{dh} \end{pmatrix} . \tag{26}$$

Using RK4, discretizing the temporal derivative reads

$$\begin{pmatrix} \mathtt{u}^{n+1} \\ \mathtt{v}^{n+1} \\ \mathtt{h}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathtt{u}^{n} \\ \mathtt{v}^{n} \\ \mathtt{h}^{n} \end{pmatrix} + \frac{\Delta t}{6} \begin{pmatrix} \mathtt{k}_1^u + 2\mathtt{k}_2^u + 2\mathtt{k}_3^u + \mathtt{k}_4^u \\ \mathtt{k}_1^v + 2\mathtt{k}_2^v + 2\mathtt{k}_3^v + \mathtt{k}_4^h \\ \mathtt{k}_1^h + 2\mathtt{k}_2^h + 2\mathtt{k}_3^h + \mathtt{k}_4^h \end{pmatrix} \tag{27}$$

with the superscript $n, n+1$ denoting the current and next time step, respectively, that lie time Δt apart. The choice of Δt is discussed below. $(\mathtt{k}^u, \mathtt{k}^v, \mathtt{k}^h)$ are approximations for $(\partial_t \mathtt{u}, \partial_t \mathtt{v}, \partial_t \mathtt{k})$ and defined as

$$(\mathtt{k}_1^u, \mathtt{k}_1^v, \mathtt{k}_1^h) = \text{rhs}(\mathtt{u}^n, \mathtt{v}^n, \mathtt{h}^n) \tag{28a}$$

$$(\mathtt{k}_2^u, \mathtt{k}_2^v, \mathtt{k}_2^h) = \text{rhs}(\mathtt{u}^n + \tfrac{\Delta t}{2}\mathtt{k}_1^u, \mathtt{v}^n + \tfrac{\Delta t}{2}\mathtt{k}_1^v, \mathtt{h}^n + \tfrac{\Delta t}{2}\mathtt{k}_1^h) \tag{28b}$$

$$(\mathtt{k}_3^u, \mathtt{k}_3^v, \mathtt{k}_3^h) = \text{rhs}(\mathtt{u}^n + \tfrac{\Delta t}{2}\mathtt{k}_2^u, \mathtt{v}^n + \tfrac{\Delta t}{2}\mathtt{k}_2^v, \mathtt{h}^n + \tfrac{\Delta t}{2}\mathtt{k}_2^h) \tag{28c}$$

$$(\mathtt{k}_4^u, \mathtt{k}_4^v, \mathtt{k}_4^h) = \text{rhs}(\mathtt{u}^n + \Delta t \mathtt{k}_3^u, \mathtt{v}^n + \Delta t \mathtt{k}_3^v, \mathtt{h}^n + \Delta t \mathtt{k}_3^h) \tag{28d}$$

3.1.1 Model code

Relevant scripts `.py` and functions, denoted with (), and dependencies ←

`swm_integration.py`

↳ `time_integration()` ← `RK3()` or `RK4()`

↳ `RK3()` ← `rhs()`

↳ `RK4()` ← `rhs()`

`swm_rhs.py`

↳ `rhs()`

3.2 Adams-Bashforth

The shallow water model code also allows to use Adams-Bashforth (AB) methods of order 1 to 5. As AB methods are multi-step methods, in this case, order 1 (i.e. Euler forward) is used for the first time step, order 2 is used for the second time step, and so on until the desired order is reached. However, this method is not discussed further here, as it was found to perform not better than RK4, as explained in the next section.

3.2.1 Model code

Relevant scripts `.py` and functions, denoted with (), and dependencies ←

`swm_integration.py`

↳ `time_integration()` ← `rhs()`, `ABcoefficients()`

↳ `ABcoefficients()`

`swm_rhs.py`

↳ `rhs()`

3.3 Choosing the time-step size Δt

In the shallow water model, the fastest propagating signals are gravity waves. The phase speed c_p of those waves is

$$c_p = \sqrt{gh} \approx \sqrt{gH} \tag{29}$$

where the approximation holds in the barotropic case where $\eta \ll H$. The CFL-number ϵ (named after Courant, Friedrichs and Lewy) is then

$$\epsilon = \frac{c_p \Delta t}{\Delta x} \tag{30}$$

Hence, choosing the CFL-number ϵ, we obtain the time step Δt as

$$\Delta t = \epsilon \frac{\Delta x}{c_p} \tag{31}$$

Using RK4, a choice of $\epsilon \leq 0.9$ was found to be stable. Multi-step schemes such as AB have the advantage, that they only require one evaluation of the right-hand side per time step (in contrast to RK4 which requires 4 evaluations of the right-hand side), which could theoretically decrease the computational time required in order to integrate the model forward. However, in practice, the 3rd order Adams-Bashforth method was found to be only stable for $\epsilon \leq 0.2$ (i.e. a decrease of Δt by a factor of 4 to 5), which means that the effective computational performance is on the same order but slightly outperformed by RK4.

3.3.1 Model code

Relevant scripts `.py` and functions, denoted with (), and dependencies $\leftarrow$

`swm_param.py`

↳ `set_param()` $\leftarrow$ `set_grid()`, `set_timestep()`

↳ `set_grid()`

↳ `set_timestep()`