交错网格与完全匹配层

Tche L.*

2018 年 3 月 1 日

Version: 2.2

有限差分法是对介质模型，也就是对计算区域先进行离散网格化，将描述介质中传播的波动微分方程，利用微商和差商的近似关系，直接化为有限差分方程来求解，模拟波的传播。

地震勘探中的有限差分根据域的不同可分为时域有限差分和频域有限差分。根据网格不同可分为同位网格[1]、交错网格[2] [3]、旋转网格[4] [5] 等。对于边界反射波的处理，有早期的旁轴近似吸收边界条件[6]、指数型吸收边界条件[7] 和现在比较流行的完全匹配层吸收边界[8]。本文主要介绍了时域交错网格有限差分方法与完全匹配层吸收边界条件，并对简单的二维声波方程和弹性波方程给出了差分格式。

* USTC 在读博士，tcheliu@mail.ustc.edu.cn

一、什么是交错网格

交错网格是将不同的地震波场分量定义在整网格点和半网格点上，合理地安排地震波场分量在网格上的相对位置,可以方便地求取所需分量的差分。同时,它将波场分裂为 x 和 z 方向上的两个分量，将二阶位移微分方程分裂为若干个一阶速度—应力方程对波场进行求解。在交错网格中，假设 u^x 和 u^z 分别定义在 x 和 z 方向的半网格点上，则它们对 x 和 z 方向的中心差分格式为[9]

$$\begin{cases} L_x(u^x_{i,j}) = \dfrac{1}{\Delta x}\displaystyle\sum_{n=1}^{N} C_n^{(N)}(u^x_{i+\frac{2n-1}{2},j} - u^x_{i-\frac{2n-1}{2},j}) \\ L_z(u^z_{i,j}) = \dfrac{1}{\Delta z}\displaystyle\sum_{n=1}^{N} C_n^{(N)}(u^z_{i,j+\frac{2n-1}{2}} - u^z_{i,j-\frac{2n-1}{2}}) \end{cases} \tag{1}$$

其中，u 为地震波场值，u^x 和 u^z 为它的两个方向分量，Δx 和 Δz 为 x 和 z 方向的空间间隔，$C_n^{(N)}$ 为差分系数，$2N$ 为差分的空间阶数。

二、什么是完全匹配层

吸收边界条件的思想就是在需要计算场值的区域之外加上一定厚度的吸收边界层，当波运行到计算边界时候，不会发生反射，而是直接穿透边界进入所加的吸收边界层，对吸收边界层设置一定的参数，从而起到吸收超出边界的波的作用。

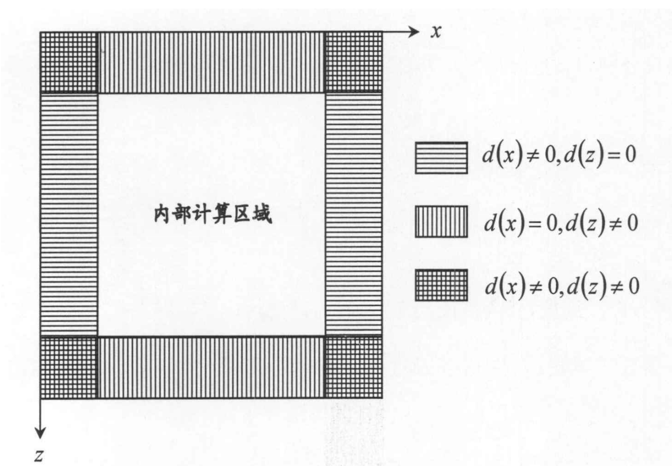

图 1: 完全匹配层吸收边示意图（图中：$d(x) = d_x$，$d(z) = d_z$）

在时域有限差分方法波场模拟中,完全匹配层(PML)吸收边界条件将波场分量在吸收边界区域分裂，分别对各个分裂的波场分量赋以不同的耗损。在计算区域截断边界外，PML 层是一种非物理的特殊吸收介质，该层的波阻抗与相邻介质的波阻抗完全匹配，因而入射波将无反射地穿过界面进行 PML 层，同时，由于 PML 层为有耗介质，进入 PML 层的入射波将迅速衰减，最终实现消弱边界反射的效果。

PML 吸收边界具体做法如图 1 所示，在内部计算区域，采用一般的速度—应力方程，而在 PML 层区域内，在频率空间域对方程中的 x 和 z 方向偏导分别作如下替换：

$$\frac{\partial}{\partial x} \longrightarrow \frac{i\omega}{i\omega + d_x}\frac{\partial}{\partial x}, \qquad \frac{\partial}{\partial z} \longrightarrow \frac{i\omega}{i\omega + d_z}\frac{\partial}{\partial z}$$

其中，ω 为角频率，d_x 和 d_z 分别为 x 和 z 方向的阻尼因子。

例如，对于如下方程：

$$\frac{\partial u}{\partial t} = A\frac{\partial v}{\partial x}$$

在内部计算区域，我们采用上式求解即可。而在 PML 层内，我们应对方程作一些调整。上式对应的频率空间域方程为：

$$i\omega u = A\frac{\partial v}{\partial x}$$

在 PML 层内，对 x 方向偏导进行替换，得到如下方程：

$$i\omega u = A\frac{i\omega}{i\omega + d_x}\frac{\partial v}{\partial x}，\text{也即 } (i\omega + d_x)u = A\frac{\partial v}{\partial x}$$

其在时间空间域的表达形式为：

$$\Big(\frac{\partial}{\partial t} + d_x\Big)u = A\frac{\partial v}{\partial x} \tag{2}$$

因此，我们在 PML 层内可采用上式求解，即可实现 PML 吸收边界层内衰减。

在对上式左侧采用差分近似的实际过程中，我们有两种近似方案。先假设上式右侧经空间差分近似后的结果为

$$A\frac{\partial v_k}{\partial x} \approx \spadesuit|_k$$

其中 ∂v_k 为 $k\Delta t$ 时刻 v 的偏导，Δt 为时间步长。同时假设 u 定义在半时间网格点上，则第一种近似方案为：

$$\Big(\frac{\partial}{\partial t} + d_x\Big)u_k = \frac{\partial u_k}{\partial t} + d_x \cdot u_k = \frac{u_{k+1/2} - u_{k-1/2}}{\Delta t} + d_x \cdot \frac{u_{k+1/2} + u_{k-1/2}}{2}$$

将上式代入式 (2)，最终，我们得到第一种近似下的时间递推关系式为：

$$u_{k+1/2} = \frac{2 - \Delta t \cdot d_x}{2 + \Delta t \cdot d_x} \cdot u_{k-1/2} + \frac{2\Delta t}{2 + \Delta t \cdot d_x} \cdot \spadesuit|_k \tag{3}$$

第二种近似方案为：

$$\Big(\frac{\partial}{\partial t} + d_x\Big)u_k = \frac{\partial u_k}{\partial t} + d_x \cdot u_k = \frac{u_{k+1/2} - u_{k-1/2}}{\Delta t} + d_x \cdot u_{k-1/2}$$

将其代入式 (2)，最终，我们得到第二种近似下的时间递推关系式为：

$$u_{k+1/2} = \big(1 - \Delta t \cdot d_x\big)u_{k-1/2} + \Delta t \cdot \spadesuit|_k \tag{4}$$

其实，我们可以将内部计算区域和 PML 层区域的方程统一起来，当 $d_x = d_z = 0$ 时 PML 层区域的方程转化为内部计算区域的方程，编程时我们可以考虑统一采用 PML 层区域的方程形式求解，只需特别地在内部计算区域令 $d_x = d_z = 0$ 即可。

那么，衰减因子 d_x 和 d_z 如何给定？对于上边界或左边界，文献[10] 给出了形如下式的衰减因子：

$$d_*(i) = d_{0*}\Big(\frac{i}{n_{pml*}}\Big)^p$$

其中，$*$ 表示 x 或 z，i 为从内部有效计算区域边界起算的 PML 层数，n_{pml*} 为在 $*$ 方向上所加载的单边 PML 层网格点数，典型地 p 的取值范围为 $1\sim4$。另外，

$$d_{0*} = \log\Big(\frac{1}{R}\Big)\frac{\tau V_s}{n_{pml*}\Delta *}$$

或

$$d_{0*} = \frac{\tau V_s}{\Delta *}\big(c_1 + c_2 n_{pml*} + c_3 n_{pml*}^2\big)$$

其中，R 为理论反射系数；τ 为微调参数，取值范围为 $3\sim4$；V_s 为横波波速；$\Delta *$ 为在 $*$ 方向上的网格间距；c_i 为多项式系数。对于 R 或 c_i 的取值如下：

$$\begin{cases} R = 0.01, & \text{当 } n_{pml*} = 5 \\ R = 0.001, & \text{当 } n_{pml*} = 10 \\ R = 0.0001, & \text{当 } n_{pml*} = 20 \end{cases} \qquad \text{或} \qquad \begin{cases} c_1 = \dfrac{8}{15} \\ c_2 = \dfrac{-3}{100} \\ c_3 = \dfrac{1}{1500} \end{cases}$$

三、空间上任意偶数阶差分近似

在交错网格方法中，波场分量的导数是在相应的分量网格节点之间的半程上计算的。因此，我们可以用下式计算方程中的一阶空间导数：

$$\frac{\partial u}{\partial x} = \frac{1}{\Delta x}\sum_{n=1}^{N}\Big\{C_n^{(N)}\big[u\big(x+\frac{2n-1}{2}\Delta x\big) - u\big(x-\frac{2n-1}{2}\Delta x\big)\big]\Big\} + O\big(\Delta x^{2N}\big) \tag{5}$$

上式中待定系数 $C_n^{(N)}$ 的准确求取是确保一阶空间导数的 $2N$ 阶差分精度的关键。将 $u\big(x+\frac{2n-1}{2}\Delta x\big)$ 和 $u\big(x-\frac{2n-1}{x}\Delta x\big)$ 在 x 处 Taylor 展开后可以发现，通过求解下列方程组即可确定待定系数 $C_n^{(N)}$：

$$\begin{bmatrix} 1^1 & 3^1 & 5^1 & \cdots & (2N-1)^1 \\ 1^3 & 3^3 & 5^3 & \cdots & (2N-1)^3 \\ 1^5 & 3^5 & 5^5 & \cdots & (2N-1)^5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1^{2N-1} & 3^{2N-1} & 5^{2N-1} & \cdots & (2N-1)^{2N-1} \end{bmatrix} \begin{bmatrix} C_1^{(N)} \\ C_2^{(N)} \\ C_3^{(N)} \\ \vdots \\ C_N^{(N)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

其解为：

$$C_m^{(N)} = \frac{(-1)^{m+1}\prod\limits_{i=1,i\neq m}^{N}(2i-1)^2}{(2m-1)\prod\limits_{i=1,i\neq m}^{N}\big|(2m-1)^2-(2i-1)^2\big|} \tag{6}$$

四、交错网格中的声波方程

如我们所常见的，在各向同性介质中，二维声波波动方程可表示为：

$$\frac{\partial^2 P}{\partial t^2} = v_P^2\Big(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial z^2}\Big) \tag{7}$$

其中，P 为压力波场或位移波场，v_P 为介质声波波速。

在交错网格中，我们将不同的波场分量定义在不同的网格点上，这就需要我们采用多波场分量的方程来进行波场模拟。在各向同性介质中，二维声波一阶速度—应力方程可表示为：

$$\begin{cases} \dfrac{\partial P}{\partial t} = -\rho v_P^2(\dfrac{\partial v_x}{\partial x} + \dfrac{\partial v_z}{\partial z}) \\ \dfrac{\partial v_x}{\partial t} = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial x} \\ \dfrac{\partial v_z}{\partial t} = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial z} \end{cases} \tag{8}$$

其中，v_x 和 v_z 分别为在 x 和 z 方向的质点运动速度波场分量，ρ 为介质密度。

在方程 (8) 中的第一个等式两边同时对 t 求偏导，交换等式右侧对时间求导和对空间求导的先后顺序，再结合方程 (8) 中的后两个等式，即可得到如式 (7) 所示的波动方程。

在 PML 吸收边界中，我们在 x 和 z 方向上采取不同的阻尼衰减因子，由于方程 (8) 中的第一个等式同时包含了对 x 和 z 方向的偏导，因此，还需要对该式作进一步拆分：

$$\begin{cases} P = P_x + P_z \\ \dfrac{\partial P_x}{\partial t} = -\rho v_P^2\dfrac{\partial v_x}{\partial x} \\ \dfrac{\partial P_z}{\partial t} = -\rho v_P^2\dfrac{\partial v_z}{\partial z} \end{cases} \tag{9}$$

其中，P_x 和 P_z 分别为应力波场 P 在 x 和 z 方向上的分量。

根据式 (8) 和 (9)，引入 PML 吸收边界条件，得到：

$$\begin{cases} (\dfrac{\partial}{\partial t} + d_x)v_x = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial x} \\ (\dfrac{\partial}{\partial t} + d_z)v_z = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial z} \\ (\dfrac{\partial}{\partial t} + d_x)P_x = -\rho v_P^2\dfrac{\partial v_x}{\partial x} \\ (\dfrac{\partial}{\partial t} + d_z)P_z = -\rho v_P^2\dfrac{\partial v_z}{\partial z} \end{cases} \tag{10}$$

按照如图 2 所示波场分量和参数排布方式，我们在时间上采用如式 (4) 所示的递推格式，在空间上采用如式 (5) 所示的任意偶数阶差分近似，可以得到在 PML 层内采用第二种近似下的时间二阶差分精度、空间 $2N$ 阶差分精度的交错网格有限差分声波方程时间递推格

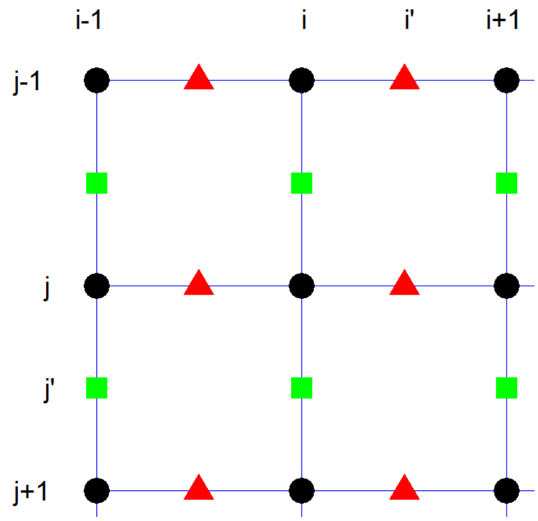

图 2: 声波交错网格示意图（●： P, ρv_P^2；▲： v_x， $^1/_\rho$；■： v_z， $^1/_\rho$）

式如下：

$$
\begin{cases}
v_x\big|_{i+1/2,j}^{k} = (1-\Delta t\cdot d_x)v_x\big|_{i+1/2,j}^{k-1} - \dfrac{\Delta t}{\rho\Delta x}\displaystyle\sum_{n=1}^{N}\Big\{C_n^{(N)}\big[P\big|_{i+1/2+(2n-1)/2,j}^{k-1/2} - P\big|_{i+1/2-(2n-1)/2,j}^{k-1/2}\big]\Big\} \\
v_z\big|_{i,j+1/2}^{k} = (1-\Delta t\cdot d_z)v_z\big|_{i,j+1/2}^{k-1} - \dfrac{\Delta t}{\rho\Delta z}\displaystyle\sum_{n=1}^{N}\Big\{C_n^{(N)}\big[P\big|_{i,j+1/2+(2n-1)/2}^{k-1/2} - P\big|_{i,j+1/2-(2n-1)/2}^{k-1/2}\big]\Big\} \\
P_x\big|_{i,j}^{k+1/2} = (1-\Delta t\cdot d_x)P_x\big|_{i,j}^{k-1/2} - \dfrac{\rho v_P^2\Delta t}{\Delta x}\displaystyle\sum_{n=1}^{N}\Big\{C_n^{(N)}\big[v_x\big|_{i+(2n-1)/2,j}^{k} - v_x\big|_{i-(2n-1)/2,j}^{k}\big]\Big\} \\
P_z\big|_{i,j}^{k+1/2} = (1-\Delta t\cdot d_z)P_z\big|_{i,j}^{k-1/2} - \dfrac{\rho v_P^2\Delta t}{\Delta z}\displaystyle\sum_{n=1}^{N}\Big\{C_n^{(N)}\big[v_z\big|_{i,j+(2n-1)/2}^{k} - v_z\big|_{i,j-(2n-1)/2}^{k}\big]\Big\}
\end{cases}
\tag{11}
$$

其中， $P = P_x + P_z$， $v_x\big|_{i+1/2,j}^{k}$ 为空间网格点 $\big((i+1/2)\Delta x, j\Delta z\big)$ 处在 $k\Delta t$ 时刻 v_x 的值， Δx 和 Δz 分别为 x 和 z 方向上空间差分步长。

五、交错网格中的弹性波方程

对于弹性波方程，如我们所常见的，在各向同性介质中，二维波动方程可表示为：

$$
\begin{cases}
\rho\dfrac{\partial^2 u_x}{\partial t^2} = \dfrac{\partial}{\partial x}\big[\lambda(\dfrac{\partial u_x}{\partial x} + \dfrac{\partial u_z}{\partial z}) + 2\mu\dfrac{\partial u_x}{\partial x}\big] + \dfrac{\partial}{\partial z}\big[\mu(\dfrac{\partial u_z}{\partial x} + \dfrac{\partial u_x}{\partial z})\big] \\
\rho\dfrac{\partial^2 u_z}{\partial t^2} = \dfrac{\partial}{\partial z}\big[\lambda(\dfrac{\partial u_x}{\partial x} + \dfrac{\partial u_z}{\partial z}) + 2\mu\dfrac{\partial u_z}{\partial z}\big] + \dfrac{\partial}{\partial x}\big[\mu(\dfrac{\partial u_z}{\partial x} + \dfrac{\partial u_x}{\partial z})\big]
\end{cases}
\tag{12}
$$

其中， u_x 和 u_z 分别为 x 和 z 方向上的位移， ρ 为介质密度， $\lambda = \rho(v_p^2 - 2v_s^2)$ 和 $\mu = \rho v_s^2$ 为介质拉梅常数， v_p 和 v_s 分别为介质的纵波速度和横波速度。

另外，我们有弹性动力学方程如下[3]：

$$
\begin{cases}
\rho\dfrac{\partial^2 u_x}{\partial t^2} = \dfrac{\partial \tau_{xx}}{\partial x} + \dfrac{\partial \tau_{xz}}{\partial z} \\
\rho\dfrac{\partial^2 u_z}{\partial t^2} = \dfrac{\partial \tau_{xz}}{\partial x} + \dfrac{\partial \tau_{zz}}{\partial z} \\
\tau_{xx} = (\lambda + 2\mu)\dfrac{\partial u_x}{\partial x} + \lambda\dfrac{\partial u_z}{\partial z} \\
\tau_{zz} = (\lambda + 2\mu)\dfrac{\partial u_z}{\partial z} + \lambda\dfrac{\partial u_x}{\partial x} \\
\tau_{xz} = \mu(\dfrac{\partial u_x}{\partial z} + \dfrac{\partial u_z}{\partial x})
\end{cases}
\tag{13}
$$

其中，$(\tau_{xx}, \tau_{zz}, \tau_{xz})$ 为应力张量。不难发现，我们将方程 (13) 的后三个等式代入前两个等式中，即可得到如式 (12) 所示的波动方程。

然而，仅有上式，由于含有对时间的二阶偏导项，我们并不能将 PML 吸收边界条件直接引进来。我们将质点运动速度波场分量 $v_x = {\partial u_x}/{\partial t}$ 和 $v_z = {\partial u_z}/{\partial t}$ 引入上式，得到如下二维弹性波一阶速度—应力方程：

$$
\begin{cases}
\dfrac{\partial v_x}{\partial t} = \dfrac{1}{\rho}(\dfrac{\partial \tau_{xx}}{\partial x} + \dfrac{\partial \tau_{xz}}{\partial z}) \\
\dfrac{\partial v_z}{\partial t} = \dfrac{1}{\rho}(\dfrac{\partial \tau_{xz}}{\partial x} + \dfrac{\partial \tau_{zz}}{\partial z}) \\
\dfrac{\partial \tau_{xx}}{\partial t} = (\lambda + 2\mu)\dfrac{\partial v_x}{\partial x} + \lambda\dfrac{\partial v_z}{\partial z} \\
\dfrac{\partial \tau_{zz}}{\partial t} = (\lambda + 2\mu)\dfrac{\partial v_z}{\partial z} + \lambda\dfrac{\partial v_x}{\partial x} \\
\dfrac{\partial \tau_{xz}}{\partial t} = \mu(\dfrac{\partial v_x}{\partial z} + \dfrac{\partial v_z}{\partial x})
\end{cases}
\tag{14}
$$

但是,由于上式的每一个等式中都同时含有对 x 和 z 的偏导,依然不能直接引入 PML 边界条件。接下来，我们需要对上式中的每一个等式作如式 (9) 所示的拆分，进一步得到：

$$
\begin{cases}
v_x = v_x^x + v_x^z, & \dfrac{\partial v_x^x}{\partial t} = \dfrac{1}{\rho}\dfrac{\partial \tau_{xx}}{\partial x}, & \dfrac{\partial v_x^z}{\partial t} = \dfrac{1}{\rho}\dfrac{\partial \tau_{xz}}{\partial z} \\
v_z = v_z^x + v_z^z, & \dfrac{\partial v_z^x}{\partial t} = \dfrac{1}{\rho}\dfrac{\partial \tau_{xz}}{\partial x}, & \dfrac{\partial v_z^z}{\partial t} = \dfrac{1}{\rho}\dfrac{\partial \tau_{zz}}{\partial z} \\
\tau_{xx} = \tau_{xx}^x + \tau_{xx}^z, & \dfrac{\partial \tau_{xx}^x}{\partial t} = (\lambda + 2\mu)\dfrac{\partial v_x}{\partial x}, & \dfrac{\partial \tau_{xx}^z}{\partial t} = \lambda\dfrac{\partial v_z}{\partial z} \\
\tau_{zz} = \tau_{zz}^x + \tau_{zz}^z, & \dfrac{\partial \tau_{zz}^x}{\partial t} = \lambda\dfrac{\partial v_x}{\partial x}, & \dfrac{\partial \tau_{zz}^z}{\partial t} = (\lambda + 2\mu)\dfrac{\partial v_z}{\partial z} \\
\tau_{xz} = \tau_{xz}^x + \tau_{xz}^z, & \dfrac{\partial \tau_{xz}^x}{\partial t} = \mu\dfrac{\partial v_z}{\partial x}, & \dfrac{\partial \tau_{xz}^z}{\partial t} = \mu\dfrac{\partial v_x}{\partial z}
\end{cases}
\tag{15}
$$

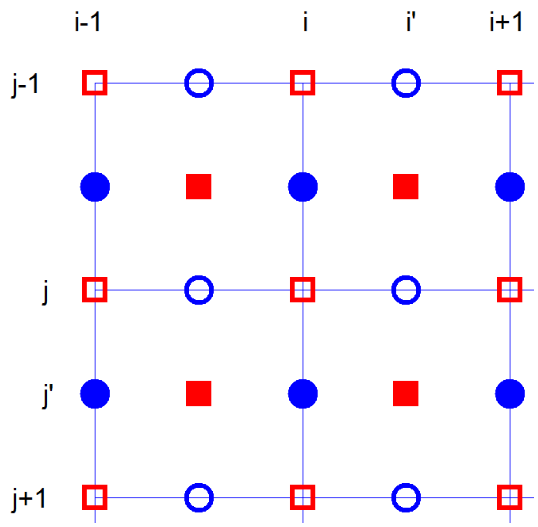

图 3: 弹性波交错网格示意图（□：v_x，$1/\rho$；■：v_z，$1/\rho$；○：τ_{xx}，τ_{zz}，$(\lambda+2\mu)$，λ；●：τ_{xz}，μ）

至此，我们可以在上式的时间微分中引入 PML 层吸收边界，得到：

$$\begin{cases}
(\frac{\partial}{\partial t}+d_x)v_x^x=\frac{1}{\rho}\frac{\partial\tau_{xx}}{\partial x}, & (\frac{\partial}{\partial t}+d_z)v_x^z=\frac{1}{\rho}\frac{\partial\tau_{xz}}{\partial z}\\
(\frac{\partial}{\partial t}+d_x)v_z^x=\frac{1}{\rho}\frac{\partial\tau_{xz}}{\partial x}, & (\frac{\partial}{\partial t}+d_z)v_z^z=\frac{1}{\rho}\frac{\partial\tau_{zz}}{\partial z}\\
(\frac{\partial}{\partial t}+d_x)\tau_{xx}^x=(\lambda+2\mu)\frac{\partial v_x}{\partial x}, & (\frac{\partial}{\partial t}+d_z)\tau_{xx}^z=\lambda\frac{\partial v_z}{\partial z}\\
(\frac{\partial}{\partial t}+d_x)\tau_{zz}^x=\lambda\frac{\partial v_x}{\partial x}, & (\frac{\partial}{\partial t}+d_z)\tau_{zz}^z=(\lambda+2\mu)\frac{\partial v_z}{\partial z}\\
(\frac{\partial}{\partial t}+d_x)\tau_{xz}^x=\mu\frac{\partial v_z}{\partial x}, & (\frac{\partial}{\partial t}+d_z)\tau_{xz}^z=\mu\frac{\partial v_x}{\partial z}
\end{cases} \tag{16}$$

其中，

$$\begin{cases}
v_x=v_x^x+v_x^z\\
v_z=v_z^x+v_z^z\\
\tau_{xx}=\tau_{xx}^x+\tau_{xx}^z\\
\tau_{zz}=\tau_{zz}^x+\tau_{zz}^z\\
\tau_{xz}=\tau_{xz}^x+\tau_{xz}^z
\end{cases} \tag{17}$$

按照如图 3 所示波场分量和参数排布方式，我们在时间上采用如式 (4) 所示的递推格式，在空间上采用如式 (5) 所示的任意偶数阶差分近似，可以得到在 PML 层内采用第二种近似下的时间二阶差分精度、空间 $2N$ 阶差分精度的交错网格有限差分弹性波方程时间递推

格式如下：

$$
\begin{cases}
v_x^x\big|_{i,j}^{k+1/2} = \big(1 - \Delta t \cdot d_x\big) v_x^x\big|_{i,j}^{k-1/2} + \dfrac{\Delta t}{\rho \Delta x} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[\tau_{xx}\big|_{i+(2n-1)/2,j}^{k} - \tau_{xx}\big|_{i-(2n-1)/2,j}^{k} \big] \Big\} \\
v_x^z\big|_{i,j}^{k+1/2} = \big(1 - \Delta t \cdot d_z\big) v_x^z\big|_{i,j}^{k-1/2} + \dfrac{\Delta t}{\rho \Delta z} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[\tau_{xz}\big|_{i,j+(2n-1)/2}^{k} - \tau_{xz}\big|_{i,j-(2n-1)/2}^{k} \big] \Big\} \\
v_z^x\big|_{i+1/2,j+1/2}^{k+1/2} = \big(1 - \Delta t \cdot d_x\big) v_z^x\big|_{i+1/2,j+1/2}^{k-1/2} + \dfrac{\Delta t}{\rho \Delta x} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[\tau_{xz}\big|_{i+1/2+(2n-1)/2,j+1/2}^{k} - \\
\qquad\qquad \tau_{xz}\big|_{i+1/2-(2n-1)/2,j+1/2}^{k} \big] \Big\} \\
v_z^z\big|_{i+1/2,j+1/2}^{k+1/2} = \big(1 - \Delta t \cdot d_z\big) v_z^z\big|_{i+1/2,j+1/2}^{k-1/2} + \dfrac{\Delta t}{\rho \Delta z} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[\tau_{zz}\big|_{i+1/2,j+1/2+(2n-1)/2}^{k} - \\
\qquad\qquad \tau_{zz}\big|_{i+1/2,j+1/2-(2n-1)/2}^{k} \big] \Big\} \\
\tau_{xx}^x\big|_{i+1/2,j}^{k+1} = \big(1 - \Delta t \cdot d_x\big) \tau_{xx}^x\big|_{i+1/2,j}^{k} + \dfrac{(\lambda+2\mu)\Delta t}{\Delta x} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_x\big|_{i+1/2+(2n-1)/2,j}^{k+1/2} - \\
\qquad\qquad v_x\big|_{i+1/2-(2n-1)/2,j}^{k+1/2} \big] \Big\} \\
\tau_{xx}^z\big|_{i+1/2,j}^{k+1} = \big(1 - \Delta t \cdot d_z\big) \tau_{xx}^z\big|_{i+1/2,j}^{k} + \dfrac{\lambda\Delta t}{\Delta z} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_z\big|_{i+1/2,j+(2n-1)/2}^{k+1/2} - \\
\qquad\qquad v_z\big|_{i+1/2,j-(2n-1)/2}^{k+1/2} \big] \Big\} \\
\tau_{zz}^x\big|_{i+1/2,j}^{k+1} = \big(1 - \Delta t \cdot d_x\big) \tau_{zz}^x\big|_{i+1/2,j}^{k} + \dfrac{\lambda\Delta t}{\Delta x} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_x\big|_{i+1/2+(2n-1)/2,j}^{k+1/2} - \\
\qquad\qquad v_x\big|_{i+1/2-(2n-1)/2,j}^{k+1/2} \big] \Big\} \\
\tau_{zz}^z\big|_{i+1/2,j}^{k+1} = \big(1 - \Delta t \cdot d_z\big) \tau_{zz}^z\big|_{i+1/2,j}^{k} + \dfrac{(\lambda+2\mu)\Delta t}{\Delta z} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_z\big|_{i+1/2,j+(2n-1)/2}^{k+1/2} - \\
\qquad\qquad v_z\big|_{i+1/2,j-(2n-1)/2}^{k+1/2} \big] \Big\} \\
\tau_{xz}^x\big|_{i,j+1/2}^{k+1} = \big(1 - \Delta t \cdot d_x\big) \tau_{xz}^x\big|_{i,j+1/2}^{k} + \dfrac{\mu\Delta t}{\Delta x} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_z\big|_{i+(2n-1)/2,j+1/2}^{k+1/2} - \\
\qquad\qquad v_z\big|_{i-(2n-1)/2,j+1/2}^{k+1/2} \big] \Big\} \\
\tau_{xz}^z\big|_{i,j+1/2}^{k+1} = \big(1 - \Delta t \cdot d_z\big) \tau_{xz}^z\big|_{i,j+1/2}^{k} + \dfrac{\mu\Delta t}{\Delta z} \displaystyle\sum_{n=1}^{N} \Big\{ C_n^{(N)} \big[v_x\big|_{i,j+1/2+(2n-1)/2}^{k+1/2} - \\
\qquad\qquad v_z\big|_{i,j+1/2-(2n-1)/2}^{k+1/2} \big] \Big\}
\end{cases}
\tag{18}
$$

其中，各波场分量之间还包含如式 (17) 所示关系，$v_x^x\big|_{i,j}^{k+1/2}$ 为空间网格点 $(i\Delta x, j\Delta z)$ 处在 $(k+1/2)\Delta t$ 时刻 v_x^x 的值。

参考文献

[1] Alterman Z., Karal F. C., 1968. Propagation of elastic waves in layered media by finite difference methods[J]. Bulletin of the Seismological Society of America, 58(1), 367-398.

[2] Virieux J., 1984. SH-wave propagation in heterogeneous media: velocity-stress finite-difference method[J]. Geophysics, 49(11), 1933-1942.

[3] Virieux J., 1986. P-SV wave propagation in heterogeneous media: Velocity-stress finite-difference method[J]. Geophysics, 51(4), 889-901.

[4] Saenger E. H., Gold N., Shapiro S. A., 2000. Modeling the propagation of elastic waves using a modified finite-difference grid[J]. Wave Motion, 31(1), 77-92.

[5] Saenger E. H., Bohlen T., 2004. Finite-difference modeling of viscoelastic and anisotropic wave propagation using the rotated staggered grid[J]. Geophysics, 69(2), 583-591.

[6] Clayton R., Engquist B., 1977. Absorbing boundary conditions for acoustic and elastic wave equations[J]. Bulletin of the Seismological Society of America, 67(6), 1529-1540.

[7] Cerjan C., Kosloff D., Kosloff R., Reshef M., 1985. A nonreflecting boundary condition for discrete acoustic and elastic wave equations[J]. Geophysics, 50(4), 705-708.

[8] Collino F., Tsogka C., 2001. Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media[J]. Geophysics, 66(1), 294-307.

[9] 孙耀充，张延腾，白超英，2013. 二维弹性及粘弹性 TTI 介质中地震波场数值模拟：四种不同网格高阶有限差分算法研究 [J]. 地球物理学进展，28(4)，1817-1827.

[10] Marcinkovich C., Olsen K., 2003. On the implementation of perfectly matched layers in a three-dimensional fourth-order velocity-stress finite difference scheme[J]. Journal of Geophysical Research, 108(B5).

附录 声波：TDFDAWFS2DSG

附录.1 Matlab 程序

```
function TDFDAWFS2DSG

% TDFDAWFS2DSG
% This is a program of Time Domain Finite Difference Acoustic Wave Field Simulating with 2-Dimension
     Staggered Grid.
% Written by Tche.L. from USTC, 2016.6.

clc; clear; close all;
% format long;

%% Input parameters

nx = 101;               % the number of grid nodes in x-direction.
nz = 101;               % the number of grid nodes in z-direction.
npmlz = 20;             % the number of grid nodes in top and bottom side of PML absorbing boundary.
npmlx = 20;             % the number of grid nodes in left and right side of PML absorbing boudary.
sx = 50;                % the grid node number of source position in x-direction.
sz = 50;                % the grid node number of source position in z-direction.
dx = 5;                 % the grid node interval in x-direction; Unit: m.
dz = 5;                 % the grid node interval in z-direction; Unit: m.
nt = 500;               % the number of time nodes for wave calculating.
dt = 1e-3;              % the time node interval; Unit: s.
nppw = 12;              % the node point number per wavelength for dominant frequency of Ricker
     wavelet source.
ampl = 1.0e0;           % the amplitude of source wavelet.
xrcvr = 1:3:nx;         % the grid node number in x-direction of reciver position on ground.
nodr = 3;               % half of the order number for spatial difference.

%% Determine the difference coefficients

B = [1 zeros(1,nodr - 1)]';
A = NaN*ones(nodr,nodr);
for i = 1:1:nodr
    A(i,:) = (1:2:2*nodr - 1).^(2*i - 1);
end
C = A\B;

%% Model and source

Nz = nz + 2*npmlz;
Nx = nx + 2*npmlx;

vp = 2000*ones(Nz,Nx);                                                    % the velocity of
     acoustic wave of model; Unit: m/s.
rho = 1000*ones(Nz,Nx);                                                   % the density of model;
     Unit: kg/m^3.
rho(fix(Nz/3):end,fix(Nx/2):end) = 500;
vp(fix(Nz/3):end,fix(Nx/2):end) = 1000;

f0 = min(vp(:))/(min(dx,dz)*nppw);                                        % the dominant frequency
     of source Ricker wavelet; Unit: Hz.
t0 = 1/f0;                                                                % the time shift of
     source Ricker wavelet; Unit: s; Suggest: 0.02 if fm = 50, or 0.05 if fm = 20.
t = dt*(1:1:nt);
src = (1 - 2*(pi*f0.*(t - t0)).^2).*exp( - (pi*f0*(t - t0)).^2);          % the time series of
```

```
source wavelet.
% The source wavelet formula refers to the equations (18) of Collino and Tsogka, 2001.

%% Perfectly matched layer absorbing factor

% R = 1e-6;                                                      % Recommend: $R = 1e-2, if $npmlr = 5; $R = 1e-3, if $npmlr = 10; $R = 1e-4, if $npmlr = 20.
% dpml0z = log(1/R)*3*max(vp(:))/(2*npmlz);
dpml0z = 3*max(vp(:))/dz*(8/15 - 3/100*npmlz + 1/1500*npmlz^2);
dpmlz = zeros(Nz,Nx);
dpmlz(1:npmlz,:) = (dpml0z*((npmlz: - 1:1)./npmlz).^2)'*ones(1,Nx);
dpmlz(npmlz + nz + 1:Nz,:) = dpmlz(npmlz: - 1:1,:);
dpml0x = 3*max(vp(:))/dx*(8/15 - 3/100*npmlx + 1/1500*npmlx^2);
dpmlx = zeros(Nz,Nx);
dpmlx(:,1:npmlx) = ones(Nz,1)*(dpml0x*((npmlx: - 1:1)./npmlx).^2);
dpmlx(:,npmlx + nx + 1:Nx) = dpmlx(:,npmlx: - 1:1);
% The PML formula refers to the equations (2) and (3) of Marcinkovich and Olsen, 2003.

%% Wavefield calculating

rho1 = rho;             % or = [(rho(:,1:end - 1) + rho(:,2:end))./2 (2*rho(:,end) - rho(:,end - 1))];
rho2 = rho;             % or = [(rho(1:end - 1,:) + rho(2:end,:))./2; (2*rho(end,:) - rho(end - 1,:))];

Coeffi1 = (2 - dt.*dpmlx)./(2 + dt.*dpmlx);
Coeffi2 = (2 - dt.*dpmlz)./(2 + dt.*dpmlz);
Coeffi3 = 1./rho1./dx.*(2*dt./(2 + dt.*dpmlx));
Coeffi4 = 1./rho2./dz.*(2*dt./(2 + dt.*dpmlz));
Coeffi5 = rho.*(vp.^2)./dx.*(2*dt./(2 + dt.*dpmlx));
Coeffi6 = rho.*(vp.^2)./dz.*(2*dt./(2 + dt.*dpmlz));

% ++++++++++++++++++++++++++++++++++++ approximate coeffient +++++++++++++++++++++++++++++++++++++
% Coeffi1 = 1 - dt.*dpmlx;
% Coeffi2 = 1 - dt.*dpmlz;
% Coeffi3 = 1./rho./dx.*dt;
% Coeffi4 = 1./rho./dz.*dt;
% Coeffi5 = rho.*(vp.^2)./dx.*dt;
% Coeffi6 = rho.*(vp.^2)./dz.*dt;
% -----------------------------------------------------------------------------------------------

NZ = Nz + 2*nodr;                                              % All values of the outermost some columns are set to zero to be a boundary condition: all of wavefield values beyond the left and right boundary are null.
NX = Nx + 2*nodr;                                              % All values of the outermost some rows are set to zero to be a boundary condition: all of wavefield values beyond the top and bottom boundary are null.

Znodes = nodr + 1:NZ - nodr;
Xnodes = nodr + 1:NX - nodr;
znodes = nodr + npmlz + 1:nodr + npmlz + nz;
xnodes = nodr + npmlx + 1:nodr + npmlx + nx;
nsrcz = nodr + npmlz + sz;
nsrcx = nodr + npmlx + sx;

Ut = NaN*ones(NZ,NX);                                         % the wavefield value preallocation.
Uz = zeros(NZ,NX);                                            % The initial condition: all of wavefield values are null before source excitation.
Ux = zeros(NZ,NX);                                            % The initial condition: all of wavefield values are null before source excitation.
```

```
Vz = zeros(NZ,NX);                                    % The initial condition: all of wavefield values are null before source excitation.
Vx = zeros(NZ,NX);                                    % The initial condition: all of wavefield values are null before source excitation.
Psum = NaN*ones(Nz,Nx);

U = NaN*ones(nz,nx,nt);

tic;
for it = 1:1:nt
    fprintf('The calculating time node is: it = %d\n',it);
    Ux(nsrcz,nsrcx) = Ux(nsrcz,nsrcx) + ampl*src(it)./2;
    Uz(nsrcz,nsrcx) = Uz(nsrcz,nsrcx) + ampl*src(it)./2;
    Ut(:,:) = Ux(:,:) + Uz(:,:);
    U(:,:,it) = Ut(znodes,xnodes);
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(Ut(Znodes,Xnodes + i) - Ut(Znodes,Xnodes + 1 - i));
    end
    Vx(Znodes,Xnodes) = Coeffi1.*Vx(Znodes,Xnodes) - Coeffi3.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(Ut(Znodes + i,Xnodes) - Ut(Znodes + 1 - i,Xnodes));
    end
    Vz(Znodes,Xnodes) = Coeffi2.*Vz(Znodes,Xnodes) - Coeffi4.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(Vx(Znodes,Xnodes - 1 + i) - Vx(Znodes,Xnodes - i));
    end
    Ux(Znodes,Xnodes) = Coeffi1.*Ux(Znodes,Xnodes) - Coeffi5.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(Vz(Znodes - 1 + i,Xnodes) - Vz(Znodes - i,Xnodes));
    end
    Uz(Znodes,Xnodes) = Coeffi2.*Uz(Znodes,Xnodes) - Coeffi6.*Psum;
end
toc;

%% Plotting

% Wavefield Snapshot
figure;% colormap gray;
clims = [min(U(:)) max(U(:))]./5;
for it = 1:5:nt
    imagesc((0:nx - 1).*dx,(0:nz - 1).*dz,U(:,:,it));%,clims);
    set(gca,'xaxislocation','top'); axis equal; axis([0 (nx - 1)*dx 0 (nz - 1)*dz]);
    colorbar; xlabel('x distance (m)'); ylabel('z depth (m)');
    title(sprintf('the snapshot of %.1f ms',it*dt*1e3),'position',[(nx - 1)*dx/2,(nz - 1)*dz*(1 + 0.07)]);
    pause(0.01);
end

% Synthetic Seismogram
syngram(:,:) = U(1,xrcvr,:);
synmax = max(abs(syngram(:)));
rcvrintv = (xrcvr(2) - xrcvr(1))*dx;
syngram = syngram./synmax.*(rcvrintv/2);
[nsyn,~] = size(syngram);
figure; hold on;
for i = 1:1:nsyn
    plot(syngram(i,:) + (xrcvr(i) - 1)*dx,t.*1e3);
```

```
end
xlabel('x distance (m)'); ylabel('travel time (ms)');
title('Synthetic Seismogram','position',[(xrcvr(1) + xrcvr(nsyn))*dx/2,t(end)*1e3*(1 + 0.07)]);
set(gca,'xaxislocation','top');
set(gca,'YDir','reverse');
hold off;

end

%% References

% Collino and Tsogka, 2001. Geophysics, Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media.
% Marcinkovich and Olsen, 2003. Journal of Geophysical Research, On the implementation of perfectly mathced layers in a three-dimensional fourth-order velocity-stress finite difference scheme.
```

附录.2 Fortran 程序

```
MODULE InputPara

  IMPLICIT NONE

  PUBLIC
  INTEGER, PARAMETER :: nx = 101, nz = 101                  ! nx: the total number of grid nodes in x-direction; nz: the total number of grid nodes in z-direction.
  INTEGER, PARAMETER :: npmlx = 20, npmlz = 20              ! npmlx: the total number of grid nodes in top and bottom side of PML absorbing boundary; npmlz: the total number of grid nodes in left and right side of PML absorbing boundary.
  INTEGER, PARAMETER :: sx = 50, sz = 50                    ! sx: the grid node number of source position in x-direction; sz: the grid node number of source position in z-direction.
  INTEGER, PARAMETER :: dx = 5, dz = 5                      ! dx: the grid node interval in x-direction; dz: the grid node interval in z-direction; Unit: m.
  INTEGER, PARAMETER :: nt = 500                            ! the total number of time nodes for wave calculating.
  REAL   , PARAMETER :: dt = 1.0E-3                         ! the time node interval, Unit: s.
  INTEGER, PARAMETER :: nppw = 12                           ! the total node point number per wavelength for dominant frequency of Ricker wavelet source.
  REAL   , PARAMETER :: amp = 1.0E0                         ! the amplitude of source wavelet.
  INTEGER, PARAMETER :: nodr = 3                            ! half of the order number for spatial difference.
  INTEGER, PARAMETER :: irstr = 1                           ! the node ID of starting reciver point.
  INTEGER, PARAMETER :: nrintv = 3                          ! the total node number between each two adjacent recivers.
  INTEGER, PARAMETER :: itstr = 1                           ! the time node ID of the first snapshot.
  INTEGER, PARAMETER :: ntintv = 5                          ! the total time node number between each two followed snapshot.

  REAL    :: src(nt)                                        ! the time series of source wavelet.
  REAL    :: vp(nz, nx), rho(nz, nx)                        ! vp: the velocity of acoustic wave of model, Unit: m/s; rho: the density of model, Unit: kg/m^3.
  INTEGER, PARAMETER :: nrcvr = CEILING(REAL(nx)/nrintv)    ! the total number of all recivers.
  INTEGER :: xrcvr(nrcvr)                                   ! the grid node number in x-direction of reciver position on ground.

  INTEGER, PRIVATE :: i

  PRIVATE nppw, amp
```

```
  PRIVATE ModelVpRho, SrcWavelet

  CONTAINS
    SUBROUTINE IntlzInputPara()
      xrcvr = [ (irstr + (i - 1)*nrintv, i = 1, nrcvr) ]
      CALL ModelVpRho()
      CALL SrcWavelet()
    END SUBROUTINE IntlzInputPara
    SUBROUTINE ModelVpRho()
      ! here you can reset $vp$ and $rho$ for the model.
      vp = 2000
      vp(nz/3:nz, nx/2:nx) = 1000
      rho = 1000
      rho(nz/3:nz, nx/2:nx) = 500
    END SUBROUTINE ModelVpRho
    SUBROUTINE SrcWavelet()
      ! here you can reset $src$ for the source wavelet.
      REAL :: f0, t0, pi = 3.1415926
      REAL :: t(nt)
      f0 = MINVAL(vp)/(MIN(dx, dz)*nppw)
      t0 = 1/f0
      t = [ (i*dt, i = 1, nt) ]
      src = amp*(1 - 2*(pi*f0*(t - t0))**2)*EXP( - (pi*f0*(t - t0))**2)
    END SUBROUTINE SrcWavelet

END MODULE InputPara

MODULE WaveExtrp

  USE InputPara
  IMPLICIT NONE

  PRIVATE
  REAL :: C(nodr)                                        ! the difference coefficients of spatial the $2*nodr$-th order difference approximating.
  INTEGER, PARAMETER :: Nzz = nz + 2*npmlz, Nxx = nx + 2*npmlx  ! Nzz: the total number of grid nodes in z-direction of compute-updating zone including PML layer; Nxx: the total number of grid nodes in x-direction of compute-updating zone including PML layer.
  REAL :: vpp(Nzz, Nxx), rhoo(Nzz, Nxx)                  ! vpp: the velocity of the expanded model including PML layer, Unit: m/s; rhoo: the density of the expanded model including PML layer, Unit: kg/m^3.
  REAL :: dpmlz(Nzz, Nxx), dpmlx(Nzz, Nxx)               ! dpmlz: the PML damping factor in z-direction; dpmlx: the PML damping factor in x-direction.
  REAL :: Coef1(Nzz, Nxx), Coef2(Nzz, Nxx), &
    & Coef3(Nzz, Nxx), Coef4(Nzz, Nxx), &
    & Coef5(Nzz, Nxx), Coef6(Nzz, Nxx)                   ! Coef1 ~ Coef6: the coefficients of wavefield time-extrapolating formula.

  INTEGER :: i, j

  REAL, PUBLIC :: P(nz, nx, nt)                          ! the calculating wavefield component varying with time.

  PUBLIC WaveExec

  CONTAINS
    SUBROUTINE WaveExec()
      CALL CalC()
      CALL ModelExpand()
      CALL CalCoefs()
      CALL CalWave()
```

```fortran
    END SUBROUTINE WaveExec
    SUBROUTINE CalC()
      REAL :: rtemp1, rtemp2
      DO i = 1,nodr,1
        rtemp1 = 1.0
        rtemp2 = 1.0
        DO j = 1,nodr,1
          IF(j == i) CYCLE
          rtemp1 = rtemp1*((2*j - 1)**2)
          rtemp2 = rtemp2*ABS((2*i - 1)**2 - (2*j - 1)**2)
        END DO
        C(i) = ( - 1)**(i + 1)*rtemp1/((2*i - 1)*rtemp2)
      END DO
    END SUBROUTINE CalC
    SUBROUTINE ModelExpand()
      vpp = 0.0
      rhoo = 0.0
      vpp(npmlz + 1:npmlz + nz, npmlx + 1:npmlx + nx) = vp
      rhoo(npmlz + 1:npmlz + nz, npmlx + 1:npmlx + nx) = rho
      DO i = 1,npmlx,1
        vpp(:, i) = vpp(:, npmlx + 1)
        vpp(:, npmlx + nx + i) = vpp(:, npmlx + nx)
        rhoo(:, i) = rhoo(:, npmlx + 1)
        rhoo(:, npmlx + nx + i) = rhoo(:, npmlx + nx)
      END DO
      DO i = 1,npmlz,1
        vpp(i, :) = vpp(npmlz + 1, :)
        vpp(npmlz + nz + i, :) = vpp(npmlz + nz, :)
        rhoo(i, :) = rhoo(npmlz + 1, :)
        rhoo(npmlz + nz + i, :) = rhoo(npmlz + nz, :)
      END DO
    END SUBROUTINE ModelExpand
    SUBROUTINE CalDpml()
      REAL :: dpml0z, dpml0x
      dpml0z = 3*MAXVAL(vp)/dz*(8.0/15 - 3.0/100*npmlz + 1.0/1500*(npmlz**2))
      DO i = 1,npmlz,1
        dpmlz(i, :) = dpml0z*((REAL(npmlz - i + 1)/npmlz)**2)
      END DO
      dpmlz(npmlz + nz + 1:Nzz, :) = dpmlz(npmlz:1:-1, :)
      dpml0x = 3*MAXVAL(vp)/dx*(8.0/15 - 3.0/100*npmlx + 1.0/1500*(npmlx**2))
      DO i = 1,npmlx,1
        dpmlx(:, i) = dpml0x*((REAL(npmlx - i + 1)/npmlx)**2)
      END DO
      dpmlx(:, npmlx + nx + 1:Nxx) = dpmlx(:, npmlx:1:-1)
    END SUBROUTINE CalDpml
    SUBROUTINE CalCoefs()
      CALL CalDpml()
      Coef1 = (2 - dt*dpmlx)/(2 + dt*dpmlx)
      Coef2 = (2 - dt*dpmlz)/(2 + dt*dpmlz)
      Coef3 = (2*dt/(2 + dt*dpmlx))/(rhoo*dx)
      Coef4 = (2*dt/(2 + dt*dpmlz))/(rhoo*dz)
      Coef5 = (2*dt/(2 + dt*dpmlx))*(rhoo*(vpp**2)/dx)
      Coef6 = (2*dt/(2 + dt*dpmlz))*(rhoo*(vpp**2)/dz)
    END SUBROUTINE CalCoefs
    SUBROUTINE CalWave()
      INTEGER :: it
      INTEGER, PARAMETER :: Nzzz = Nzz + 2*nodr, Nxxx = Nxx + 2*nodr
      INTEGER :: znds(nz) = [ (nodr + npmlz + i, i = 1,nz,1) ], &
        & xnds(nx) = [ (nodr + npmlx + i, i = 1,nx,1) ]
      INTEGER :: Zznds(Nzz) = [ (nodr + i, i = 1,Nzz,1) ], &
        & Xxnds(Nxx) = [ (nodr + i, i = 1,Nxx,1) ]
```

```
      INTEGER :: nsrcz = nodr + npmlz + sz, nsrcx = nodr + npmlx + sx
      REAL    :: Pt(Nzzz, Nxxx) = 0, &
        & Pz(Nzzz, Nxxx) = 0, Px(Nzzz, Nxxx) = 0, &
        & vz(Nzzz, Nxxx) = 0, vx(Nzzz, Nxxx) = 0
      REAL    :: SpcSum(Nzz, Nxx)
      DO it = 1,nt,1
        WRITE(*,"(A,G0)") 'The calculating time node is: it = ',it
        Px(nsrcz, nsrcx) = Px(nsrcz, nsrcx) + src(it)/2
        Pz(nsrcz, nsrcx) = Pz(nsrcz, nsrcx) + src(it)/2
        Pt(:, :) = Px(:, :) + Pz(:, :)
        P(:, :, it) = Pt(znds, xnds)
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(Pt(Znds, Xxnds + i) - Pt(Znds, Xxnds + 1 - i))
        END DO
        vx(Znds, Xxnds) = Coef1*vx(Znds, Xxnds) - Coef3*SpcSum
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(Pt(Znds + i, Xxnds) - Pt(Znds + 1 - i, Xxnds))
        END DO
        vz(Znds, Xxnds) = Coef2*vz(Znds, Xxnds) - Coef4*SpcSum
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(vx(Znds, Xxnds - 1 + i) - vx(Znds, Xxnds - i))
        END DO
        Px(Znds, Xxnds) = Coef1*Px(Znds, Xxnds) - Coef5*SpcSum
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(vz(Znds - 1 + i, Xxnds) - vz(Znds - i, Xxnds))
        END DO
        Pz(Znds, Xxnds) = Coef2*Pz(Znds, Xxnds) - Coef6*SpcSum
      END DO
    END SUBROUTINE CalWave

END MODULE WaveExtrp

!************************  TDFDAWFS2DSG  **********************
! Time Domain Finite Difference Acoustic Wave Field Simulating with 2-Dimension Staggered Grid
! Written by Tche. L. from USTC, 2016.7.
! References:
!   Collino and Tsogka, 2001. Geophysics, Application of the perfectly matched absorbing layer model
     to the linear elastodynamic problem in anisotropic heterogeneous media.
!   Marcinkovich and Olsen, 2003. Journal of Geophysical Research, On the implementation of perfectly
      mathced layers in a three-dimensional fourth-order velocity-stress finite difference scheme.
!*************************************************************
PROGRAM TDFDAWFS2DSG

  USE InputPara
  USE WaveExtrp
  IMPLICIT NONE

  CHARACTER(LEN = 128) :: SnapFile = './data/Snapshot_****.dat'       ! the snapshot file name
     template.
  CHARACTER(LEN = 128) :: SyntFile = './data/SyntRcrd.dat'                       ! the synthetic
     record file name.
  REAL    :: SyntR(nrcvr, nt)
  INTEGER :: i

  CALL IntlzInputPara()
  CALL WaveExec()
  DO i = itstr,nt,ntintv
```

```
    WRITE(SnapFile(21:24),"(I4.4)") i
    CALL Output(TRIM(SnapFile), nz, nx, P(:, :, i))
  END DO
  DO i = 1,nt,1
    SyntR(:, i) = P(1, xrcvr, i)
  END DO
  CALL Output(TRIM(SyntFile), nrcvr, nt, SyntR)

END PROGRAM TDFDAWFS2DSG

SUBROUTINE Output(Outfile, M, N, OutA)
  IMPLICIT NONE
  CHARACTER(LEN = *), INTENT(IN) :: Outfile
  INTEGER, INTENT(IN) :: M, N
  REAL, INTENT(IN)    :: OutA(M, N)
  CHARACTER(LEN = 40) :: FmtStr
  INTEGER :: i, j
  INTEGER :: funit
  WRITE(FmtStr,"('(',G0,'E15.6)')") N
  OPEN(NEWUNIT = funit, FILE = Outfile, STATUS = 'UNKNOWN')
    DO i = 1,M,1
      WRITE(funit, FmtStr) (OutA(i, j), j = 1,N,1)
    END DO
  CLOSE(funit)
END SUBROUTINE Output
```

附录 弹性波：TDFDEWFS2DSG

附录.1 Matlab 程序

```matlab
function TDFDEWFS2DSG

% TDFDEWFS2DSG
% This is a program of Time Domain Finite Difference Elastic Wave Field Simulating with 2-Dimension Staggered Grid.
% Written by Tche.L. from USTC, 2016.7.

clc; clear; close all;
% format long;

%% Input parameters

nx = 159;               % the number of grid nodes in x-direction.
nz = 159;               % the number of grid nodes in z-direction.
npmlz = 20;             % the number of grid nodes in top and bottom side of PML absorbing boundary.
npmlx = 20;             % the number of grid nodes in left and right side of PML absorbing boudary.
sx = 80;                % the grid node number of source position in x-direction.
sz = 80;                % the grid node number of source position in z-direction.
dx = 5;                 % the grid node interval in x-direction; Unit: m.
dz = 5;                 % the grid node interval in z-direction; Unit: m.
nt = 500;               % the number of time nodes for wave calculating.
dt = 1e-3;              % the time node interval; Unit: s.
nppw = 12;              % the node point number per wavelength for dominant frequency of Ricker wavelet source.
ampl = 1.0e0;           % the amplitude of source wavelet.
xrcvr = 1:3:nx;         % the grid node number in x-direction of reciver position on ground.
nodr = 1;               % half of the order number for spatial difference.

%% Determine the difference coefficients

B = [1 zeros(1,nodr - 1)]';
A = NaN*ones(nodr,nodr);
for i = 1:1:nodr
    A(i,:) = (1:2:2*nodr - 1).^(2*i - 1);
end
C = A\B;

%% Model and source

Nz = nz + 2*npmlz;
Nx = nx + 2*npmlx;

vp = 2000*ones(Nz,Nx);                                      % the velocity of P-wave of model; Unit: m/s.
vs = 1000*ones(Nz,Nx);                                      % the velocity of S-wave of model; Unit: m/s.
rho = 1000*ones(Nz,Nx);                                     % the density of model; Unit: kg/m^3.
% vp(fix(Nz/3):end,fix(Nx/2):end) = 1500;

lmd = rho.*(vp.^2 - 2.*vs.^2);                              % the lame parameter lambda of elastic wave of model.
mu = rho.*vs.^2;                                            % the lame parameter mu of elastic wave of model.
```

```
f0 = min(vs(:))/(max(dx,dz)*nppw);                                   % the dominant frequency
     of source Ricker wavelet; Unit: Hz.
t0 = 1/f0;                                                           % the time shift of
     source Ricker wavelet; Unit: s; Suggest: 0.02 if fm = 50, or 0.05 if fm = 20.
t = dt*(1:1:nt);
src = (1 - 2*(pi*f0.*(t - t0)).^2).*exp( - (pi*f0*(t - t0)).^2);            % the time series of
     source wavelet.
% The source wavelet formula refers to the equations (18) of Collino and Tsogka, 2001.

%% Perfectly matched layer absorbing factor

% R = 1e-6;                                                           % Recommend: $R = 1e-2,
      if $npmlr = 5; $R = 1e-3, if $npmlr = 10; $R = 1e-4, if $npmlr = 20.
% dpml0z = log(1/R)*3*max(vs(:))/(2*npmlz);
dpml0z = 3*max(vs(:))/dz*(8/15 - 3/100*npmlz + 1/1500*npmlz^2);
dpmlz = zeros(Nz,Nx);
dpmlz(1:npmlz,:) = (dpml0z*((npmlz: - 1:1)./npmlz).^2)'*ones(1,Nx);
dpmlz(npmlz + nz + 1:Nz,:) = dpmlz(npmlz: - 1:1,:);
dpml0x = 3*max(vs(:))/dx*(8/15 - 3/100*npmlx + 1/1500*npmlx^2);
dpmlx = zeros(Nz,Nx);
dpmlx(:,1:npmlx) = ones(Nz,1)*(dpml0x*((npmlx: - 1:1)./npmlx).^2);
dpmlx(:,npmlx + nx + 1:Nx) = dpmlx(:,npmlx: - 1:1);
% The PML formula refers to the equations (2) and (3) of Marcinkovich and Olsen, 2003.

%% Wavefield calculating


Coeffi1 = (2 - dt.*dpmlx)./(2 + dt.*dpmlx);
Coeffi2 = (2 - dt.*dpmlz)./(2 + dt.*dpmlz);
Coeffi3 = 2*dt./(2 + dt.*dpmlx)./rho./dx;
Coeffi4 = 2*dt./(2 + dt.*dpmlz)./rho./dz;
Coeffi5 = 2*dt./(2 + dt.*dpmlx).*(lmd + 2.*mu)./dx;
Coeffi6 = 2*dt./(2 + dt.*dpmlz).*lmd./dz;
Coeffi7 = 2*dt./(2 + dt.*dpmlx).*lmd./dx;
Coeffi8 = 2*dt./(2 + dt.*dpmlz).*(lmd + 2.*mu)./dz;
Coeffi9 = 2*dt./(2 + dt.*dpmlx).*mu./dx;
Coeffi0 = 2*dt./(2 + dt.*dpmlz).*mu./dz;

% ++++++++++++++++++++++++++++++++++++ approximate coeffient +++++++++++++++++++++++++++++++++++++
% Coeffi1 = 1 - dt.*dpmlx;
% Coeffi2 = 1 - dt.*dpmlz;
% Coeffi3 = dt./rho./dx;
% Coeffi4 = dt./rho./dz;
% Coeffi5 = (lmd + 2.*mu).*dt./dx;
% Coeffi6 = lmd.*dt./dz;
% Coeffi7 = lmd.*dt./dx;
% Coeffi8 = (lmd + 2.*mu).*dt./dz;
% Coeffi9 = mu.*dt./dx;
% Coeffi0 = mu.*dt./dz;
% ------------------------------------------------------------------------------------------------

NZ = Nz + 2*nodr;
NX = Nx + 2*nodr;

Znodes = nodr + 1:NZ - nodr;
Xnodes = nodr + 1:NX - nodr;
znodes = nodr + npmlz + 1:nodr + npmlz + nz;
xnodes = nodr + npmlx + 1:nodr + npmlx + nx;
nsrcz = nodr + npmlz + sz;
nsrcx = nodr + npmlx + sx;

vxt = zeros(NZ,NX);
```

```
vxx = zeros(NZ,NX);
vxz = zeros(NZ,NX);
vzt = zeros(NZ,NX);
vzx = zeros(NZ,NX);
vzz = zeros(NZ,NX);
txxt = zeros(NZ,NX);
txxx = zeros(NZ,NX);
txxz = zeros(NZ,NX);
tzzt = zeros(NZ,NX);
tzzx = zeros(NZ,NX);
tzzz = zeros(NZ,NX);
txzt = zeros(NZ,NX);
txzx = zeros(NZ,NX);
txzz = zeros(NZ,NX);
Psum = NaN*ones(Nz,Nx);

P = NaN*ones(nz,nx,nt);

tic;
for it = 1:1:nt
    fprintf('The calculating time node is: it = %d\n',it);
    %% load source
    txxx(nsrcz,nsrcx) = txxx(nsrcz,nsrcx) + ampl*src(it)./4;
    txxz(nsrcz,nsrcx) = txxz(nsrcz,nsrcx) + ampl*src(it)./4;
    tzzx(nsrcz,nsrcx) = tzzx(nsrcz,nsrcx) + ampl*src(it)./4;
    tzzz(nsrcz,nsrcx) = tzzz(nsrcz,nsrcx) + ampl*src(it)./4;
    txxt(:,:) = txxx(:,:) + txxz(:,:);
    tzzt(:,:) = tzzx(:,:) + tzzz(:,:);
    P(:,:,it) = txxt(znodes,xnodes);
%     P(:,:,it) = tzzt(znodes,xnodes);
%     P(:,:,it) = txzt(znodes,xnodes);
    %% calculate $v_x$
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(txxt(Znodes,Xnodes + i - 1) - txxt(Znodes,Xnodes - i));
    end
    vxx(Znodes,Xnodes) = Coeffi1.*vxx(Znodes,Xnodes) + Coeffi3.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(txzt(Znodes + i - 1,Xnodes) - txzt(Znodes - i,Xnodes));
    end
    vxz(Znodes,Xnodes) = Coeffi2.*vxz(Znodes,Xnodes) + Coeffi4.*Psum;
    vxt(:,:) = vxx(:,:) + vxz(:,:);
%     P(:,:,it) = vxt(znodes,xnodes);
    %% calculate $v_z$
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(txzt(Znodes,Xnodes + i) - txzt(Znodes,Xnodes - i + 1));
    end
    vzx(Znodes,Xnodes) = Coeffi1.*vzx(Znodes,Xnodes) + Coeffi3.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(tzzt(Znodes + i,Xnodes) - tzzt(Znodes - i + 1,Xnodes));
    end
    vzz(Znodes,Xnodes) = Coeffi2.*vzz(Znodes,Xnodes) + Coeffi4.*Psum;
    vzt(:,:) = vzx(:,:) + vzz(:,:);
%     P(:,:,it) = vzt(znodes,xnodes);
    %% calculate $\tau_{xx}$ and $\tau_{zz}$
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(vxt(Znodes,Xnodes + i) - vxt(Znodes,Xnodes - i + 1));
```

```
    end
    txxx(Znodes,Xnodes) = Coeffi1.*txxx(Znodes,Xnodes) + Coeffi5.*Psum;
    tzzx(Znodes,Xnodes) = Coeffi1.*tzzx(Znodes,Xnodes) + Coeffi7.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(vzt(Znodes + i - 1,Xnodes) - vzt(Znodes - i,Xnodes));
    end
    txxz(Znodes,Xnodes) = Coeffi2.*txxz(Znodes,Xnodes) + Coeffi6.*Psum;
    tzzz(Znodes,Xnodes) = Coeffi2.*tzzz(Znodes,Xnodes) + Coeffi8.*Psum;
    txxt(:,:) = txxx(:,:) + txxz(:,:);
    tzzt(:,:) = tzzx(:,:) + tzzz(:,:);
    %% calculate $\tau_{xz}$
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(vzt(Znodes,Xnodes + i - 1) - vzt(Znodes,Xnodes - i));
    end
    txzx(Znodes,Xnodes) = Coeffi1.*txzx(Znodes,Xnodes) + Coeffi9.*Psum;
    Psum(:,:) = 0;
    for i = 1:1:nodr
        Psum = Psum + C(i).*(vxt(Znodes + i,Xnodes) - vxt(Znodes - i + 1,Xnodes));
    end
    txzz(Znodes,Xnodes) = Coeffi2.*txzz(Znodes,Xnodes) + Coeffi0.*Psum;
    txzt(:,:) = txzx(:,:) + txzz(:,:);
end
toc;

%% Plotting

% Wavefield Snapshot
figure;% colormap gray;
clims = [min(P(:)) max(P(:))]./5;
for it = 1:5:nt
    imagesc((0:nx - 1).*dx,(0:nz - 1).*dz,P(:,:,it));%,clims);
    set(gca,'xaxislocation','top'); axis equal; axis([0 (nx - 1)*dx 0 (nz - 1)*dz]);
    colorbar; xlabel('x distance (m)'); ylabel('z depth (m)');
    title(sprintf('the snapshot of %.1f ms',it*dt*1e3),'position',[(nx - 1)*dx/2,(nz - 1)*dz*(1 +
     0.07)]);
    pause(0.01);
end

% Synthetic Seismogram
syngram(:,:) = P(1,xrcvr,:);
synmax = max(abs(syngram(:)));
rcvrintv = (xrcvr(2) - xrcvr(1))*dx;
syngram = syngram./synmax.*(rcvrintv/2);
[nsyn,~] = size(syngram);
figure; hold on;
for i = 1:1:nsyn
    plot(syngram(i,:) + (xrcvr(i) - 1)*dx,t.*1e3);
end
xlabel('x distance (m)'); ylabel('travel time (ms)');
title('Synthetic Seismogram','position',[(xrcvr(1) + xrcvr(nsyn))*dx/2,t(end)*1e3*(1 + 0.07)]);
set(gca,'xaxislocation','top');
set(gca,'YDir','reverse');
hold off;

end

%% References

% Collino and Tsogka, 2001. Geophysics, Application of the perfectly matched absorbing layer model to
```

```
    the linear elastodynamic problem in anisotropic heterogeneous media.
% Marcinkovich and Olsen, 2003. Journal of Geophysical Research, On the implementation of perfectly mathced layers in a three–dimensional fourth–order velocity–stress finite difference scheme.
```

附录.2 Fortran 程序

```
MODULE InputPara

  IMPLICIT NONE

  PUBLIC
  INTEGER, PARAMETER :: nx = 160, nz = 160                    ! nx: the total number of grid nodes in x–direction; nz: the total number of grid nodes in z–direction.
  INTEGER, PARAMETER :: npmlx = 20, npmlz = 20                ! npmlx: the total number of grid nodes in top and bottom side of PML absorbing boundary; npmlz: the total number of grid nodes in left and right side of PML absorbing boundary.
  INTEGER, PARAMETER :: sx = 80, sz = 80                      ! sx: the grid node number of source position in x–direction; sz: the grid node number of source position in z–direction.
  INTEGER, PARAMETER :: dx = 5, dz = 5                        ! dx: the grid node interval in x–direction; dz: the grid node interval in z–direction; Unit: m.
  INTEGER, PARAMETER :: nt = 500                              ! the total number of time nodes for wave calculating.
  REAL   , PARAMETER :: dt = 1.0E–3                           ! the time node interval, Unit: s.
  INTEGER, PARAMETER :: nppw = 12                             ! the total node point number per wavelength for dominant frequency of Ricker wavelet source.
  REAL   , PARAMETER :: amp = 1.0E0                           ! the amplitude of source wavelet.
  INTEGER, PARAMETER :: nodr = 3                              ! half of the order number for spatial difference.
  INTEGER, PARAMETER :: irstr = 1                             ! the node ID of starting reciver point.
  INTEGER, PARAMETER :: nrintv = 3                            ! the total node number between each two adjacent recivers.
  INTEGER, PARAMETER :: itstr = 1                             ! the time node ID of the first snapshot.
  INTEGER, PARAMETER :: ntintv = 5                            ! the total time node number between each two followed snapshot.

  REAL    :: src(nt)                                          ! the time series of source wavelet.
  REAL    :: vp(nz, nx), vs(nz, nx), rho(nz, nx)              ! vp: the velocity of P–wave of model, Unit: m/s; vs: the velocity of S–wave of model, Unit: m/s; rho: the density of model, Unit: kg/m^3.
  INTEGER, PARAMETER :: nrcvr = CEILING(REAL(nx)/nrintv)      ! the total number of all recivers.
  INTEGER :: xrcvr(nrcvr)                                     ! the grid node number in x–direction of reciver position on ground.

  INTEGER, PRIVATE :: i

  PRIVATE nppw, amp
  PRIVATE ModelVpRho, SrcWavelet

  CONTAINS
    SUBROUTINE IntlzInputPara()
      xrcvr = [ (irstr + (i – 1)*nrintv, i = 1, nrcvr) ]
      CALL ModelVpRho()
      CALL SrcWavelet()
    END SUBROUTINE IntlzInputPara
    SUBROUTINE ModelVpRho()
      ! here you can reset $vp$ and $rho$ for the model.
      vp = 2000
```

```
      vs = 1000
      rho = 1000
    END SUBROUTINE ModelVpRho
    SUBROUTINE SrcWavelet()
      ! here you can reset $src$ for the source wavelet.
      REAL :: f0, t0, pi = 3.1415926
      REAL :: t(nt)
      f0 = MINVAL(vs)/(MIN(dx, dz)*nppw)
      t0 = 1/f0
      t = [ (i*dt, i = 1, nt) ]
      src = amp*(1 - 2*(pi*f0*(t - t0))**2)*EXP( - (pi*f0*(t - t0))**2)
    END SUBROUTINE SrcWavelet

END MODULE InputPara

MODULE WaveExtrp

  USE InputPara
  IMPLICIT NONE

  PRIVATE
  REAL :: C(nodr)                                             ! the difference coefficients of
     spatial the $2*nodr$-th order difference approximating.
  INTEGER, PARAMETER :: Nzz = nz + 2*npmlz, Nxx = nx + 2*npmlx  ! Nzz: the total number of grid nodes
      in z-direction of compute-updating zone including PML layer; Nxx: the total number of grid
     nodes in x-direction of compute-updating zone including PML layer.
  REAL :: vpp(Nzz, Nxx), vss(Nzz, Nxx), rhoo(Nzz, Nxx)          ! vpp: the velocity of P-wave of the
     expanded model including PML layer, Unit: m/s; vss: the velocity of S-wave of the expanded model
      including PML layer, Unit: m/s; rhoo: the density of the expanded model including PML layer,
     Unit: kg/m^3.
  REAL :: lmdd(Nzz, Nxx), muu(Nzz, Nxx)                         ! lmdd: the lame parameter lambda of
     elastic wave of the expanded model including PML layer; muu: the lame parameter mu of elastic
     wave of the expanded model including PML layer.
  REAL :: dpmlz(Nzz, Nxx), dpmlx(Nzz, Nxx)                      ! dpmlz: the PML damping factor in z-
     direction; dpmlx: the PML damping factor in x-direction.
  REAL :: Coef1(Nzz, Nxx), Coef2(Nzz, Nxx), &
    & Coef3(Nzz, Nxx), Coef4(Nzz, Nxx), &
    & Coef5(Nzz, Nxx), Coef6(Nzz, Nxx), &
    & Coef7(Nzz, Nxx), Coef8(Nzz, Nxx), &
    & Coef9(Nzz, Nxx), Coef0(Nzz, Nxx)                          ! Coef1 ~ Coef0: the coefficients of
     wavefield time-extrapolating formula.

  INTEGER :: i, j

  REAL, PUBLIC :: P(nz, nx, nt)                                 ! the calculating wavefield component
      varying with time.

  PUBLIC WaveExec

  CONTAINS
    SUBROUTINE WaveExec()
      CALL CalC()
      CALL ModelExpand()
      CALL CalCoefs()
      CALL CalWave()
    END SUBROUTINE WaveExec
    SUBROUTINE CalC()
      REAL :: rtemp1, rtemp2
      DO i = 1,nodr,1
        rtemp1 = 1.0
        rtemp2 = 1.0
```

```fortran
      DO j = 1,nodr,1
        IF(j == i) CYCLE
        rtemp1 = rtemp1*((2*j - 1)**2)
        rtemp2 = rtemp2*ABS((2*i - 1)**2 - (2*j - 1)**2)
      END DO
      C(i) = ( - 1)**(i + 1)*rtemp1/((2*i - 1)*rtemp2)
    END DO
  END SUBROUTINE CalC
  SUBROUTINE ModelExpand()
    vpp = 0.0
    vss = 0.0
    rhoo = 0.0
    vpp(npmlz + 1:npmlz + nz, npmlx + 1:npmlx + nx) = vp
    vss(npmlz + 1:npmlz + nz, npmlx + 1:npmlx + nx) = vs
    rhoo(npmlz + 1:npmlz + nz, npmlx + 1:npmlx + nx) = rho
    DO i = 1,npmlx,1
      vpp(:, i) = vpp(:, npmlx + 1)
      vpp(:, npmlx + nx + i) = vpp(:, npmlx + nx)
      vss(:, i) = vss(:, npmlx + 1)
      vss(:, npmlx + nx + i) = vss(:, npmlx + nx)
      rhoo(:, i) = rhoo(:, npmlx + 1)
      rhoo(:, npmlx + nx + i) = rhoo(:, npmlx + nx)
    END DO
    DO i = 1,npmlz,1
      vpp(i, :) = vpp(npmlz + 1, :)
      vpp(npmlz + nz + i, :) = vpp(npmlz + nz, :)
      vss(i, :) = vss(npmlz + 1, :)
      vss(npmlz + nz + i, :) = vss(npmlz + nz, :)
      rhoo(i, :) = rhoo(npmlz + 1, :)
      rhoo(npmlz + nz + i, :) = rhoo(npmlz + nz, :)
    END DO
    lmdd = rhoo*(vpp**2 - 2*(vss**2))
    muu = rhoo*(vss**2)
  END SUBROUTINE ModelExpand
  SUBROUTINE CalDpml()
    REAL :: dpml0z, dpml0x
    dpml0z = 3*MAXVAL(vs)/dz*(8.0/15 - 3.0/100*npmlz + 1.0/1500*(npmlz**2))
    DO i = 1,npmlz,1
      dpmlz(i, :) = dpml0z*((REAL(npmlz - i + 1)/npmlz)**2)
    END DO
    dpmlz(npmlz + nz + 1:Nzz, :) = dpmlz(npmlz:1:-1, :)
    dpml0x = 3*MAXVAL(vs)/dx*(8.0/15 - 3.0/100*npmlx + 1.0/1500*(npmlx**2))
    DO i = 1,npmlx,1
      dpmlx(:, i) = dpml0x*((REAL(npmlx - i + 1)/npmlx)**2)
    END DO
    dpmlx(:, npmlx + nx + 1:Nxx) = dpmlx(:, npmlx:1:-1)
  END SUBROUTINE CalDpml
  SUBROUTINE CalCoefs()
    CALL CalDpml()
    Coef1 = (2 - dt*dpmlx)/(2 + dt*dpmlx)
    Coef2 = (2 - dt*dpmlz)/(2 + dt*dpmlz)
    Coef3 = (2*dt/(2 + dt*dpmlx))/rhoo/dx
    Coef4 = (2*dt/(2 + dt*dpmlz))/rhoo/dz
    Coef5 = (2*dt/(2 + dt*dpmlx))*(lmdd + 2*muu)/dx
    Coef6 = (2*dt/(2 + dt*dpmlz))*lmdd/dz
    Coef7 = (2*dt/(2 + dt*dpmlx))*lmdd/dx
    Coef8 = (2*dt/(2 + dt*dpmlz))*(lmdd + 2*muu)/dz
    Coef9 = (2*dt/(2 + dt*dpmlx))*muu/dx
    Coef0 = (2*dt/(2 + dt*dpmlz))*muu/dz
  END SUBROUTINE CalCoefs
  SUBROUTINE CalWave()
```

```
    INTEGER :: it
    INTEGER, PARAMETER :: Nzzz = Nzz + 2*nodr, Nxxx = Nxx + 2*nodr
    INTEGER :: znds(nz) = [ (nodr + npmlz + i, i = 1,nz,1) ], &
      & xnds(nx) = [ (nodr + npmlx + i, i = 1,nx,1) ]
    INTEGER :: Zznds(Nzz) = [ (nodr + i, i = 1,Nzz,1) ], &
      & Xxnds(Nxx) = [ (nodr + i, i = 1,Nxx,1) ]
    INTEGER :: nsrcz = nodr + npmlz + sz, nsrcx = nodr + npmlx + sx
    REAL    :: vxt(Nzzz, Nxxx) = 0, vxx(Nzzz, Nxxx) = 0, &
      & vxz(Nzzz, Nxxx) = 0, vzt(Nzzz, Nxxx) = 0, &
      & vzx(Nzzz, Nxxx) = 0, vzz(Nzzz, Nxxx) = 0, &
      & txxt(Nzzz, Nxxx) = 0, txxx(Nzzz, Nxxx) = 0, &
      & txxz(Nzzz, Nxxx) = 0, tzzt(Nzzz, Nxxx) = 0, &
      & tzzx(Nzzz, Nxxx) = 0, tzzz(Nzzz, Nxxx) = 0, &
      & txzt(Nzzz, Nxxx) = 0, txzx(Nzzz, Nxxx) = 0, &
      & txzz(Nzzz, Nxxx) = 0, SpcSum(Nzz, Nxx) = 0
    DO it = 1,nt,1
      WRITE(*,"(A,G0)") 'The calculating time node is: it = ',it
      !! load source
      txxx(nsrcz, nsrcx) = txxx(nsrcz, nsrcx) + src(it)/4
      txxz(nsrcz, nsrcx) = txxz(nsrcz, nsrcx) + src(it)/4
      tzzx(nsrcz, nsrcx) = tzzx(nsrcz, nsrcx) + src(it)/4
      tzzz(nsrcz, nsrcx) = tzzz(nsrcz, nsrcx) + src(it)/4
      txxt(:, :) = txxx(:, :) + txxz(:, :)
      tzzt(:, :) = tzzx(:, :) + tzzz(:, :)
      P(:, :, it) = txxt(znds, xnds);
!       P(:, :, it) = tzzt(znds, xnds);
!       P(:, :, it) = txzt(znds, xnds);
      !! calculate $v_x$
      SpcSum = 0
      DO i = 1,nodr,1
        SpcSum = SpcSum + C(i)*(txxt(Zznds, Xxnds + i - 1) - txxt(Zznds, Xxnds - i))
      END DO
      vxx(Zznds, Xxnds) = Coef1*vxx(Zznds, Xxnds) + Coef3*SpcSum
      SpcSum = 0
      DO i = 1,nodr,1
        SpcSum = SpcSum + C(i)*(txzt(Zznds + i - 1, Xxnds) - txzt(Zznds - i, Xxnds))
      END DO
      vxz(Zznds, Xxnds) = Coef2*vxz(Zznds, Xxnds) + Coef4*SpcSum
      vxt(:, :) = vxx(:, :) + vxz(:, :)
!       P(:, :, it) = vxt(znds, xnds)
      !! calculate $v_z$
      SpcSum = 0
      DO i = 1,nodr,1
        SpcSum = SpcSum + C(i)*(txzt(Zznds, Xxnds + i) - txzt(Zznds, Xxnds - i + 1))
      END DO
      vzx(Zznds, Xxnds) = Coef1*vzx(Zznds, Xxnds) + Coef3*SpcSum
      SpcSum = 0
      DO i = 1,nodr,1
        SpcSum = SpcSum + C(i)*(tzzt(Zznds + i, Xxnds) - tzzt(Zznds - i + 1, Xxnds))
      END DO
      vzz(Zznds, Xxnds) = Coef2*vzz(Zznds, Xxnds) + Coef4*SpcSum
      vzt(:, :) = vzx(:, :) + vzz(:, :)
!       P(:, :, it) = vzt(znds, xnds)
      !! calculate $\tau_{xx}$ and $\tau_{zz}$
      SpcSum = 0
      DO i = 1,nodr,1
        SpcSum = SpcSum + C(i)*(vxt(Zznds, Xxnds + i) - vxt(Zznds, Xxnds - i + 1))
      END DO
      txxx(Zznds, Xxnds) = Coef1*txxx(Zznds, Xxnds) + Coef5*SpcSum
      tzzx(Zznds, Xxnds) = Coef1*tzzx(Zznds, Xxnds) + Coef7*SpcSum
      SpcSum = 0
```

```
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(vzt(Zznds + i - 1, Xxnds) - vzt(Zznds - i, Xxnds))
        END DO
        txxz(Zznds, Xxnds) = Coef2*txxz(Zznds, Xxnds) + Coef6*SpcSum
        tzzz(Zznds, Xxnds) = Coef1*tzzz(Zznds, Xxnds) + Coef8*SpcSum
        txxt(:, :) = txxx(:, :) + txxz(:, :)
        tzzt(:, :) = tzzx(:, :) + tzzz(:, :)
        !! calculate $\tau_{xz}$
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(vzt(Zznds, Xxnds + i - 1) - vzt(Zznds, Xxnds - i))
        END DO
        txzx(Zznds, Xxnds) = Coef1*txzx(Zznds, Xxnds) + Coef9*SpcSum
        SpcSum = 0
        DO i = 1,nodr,1
          SpcSum = SpcSum + C(i)*(vxt(Zznds + i, Xxnds) - vxt(Zznds - i + 1, Xxnds))
        END DO
        txzz(Zznds, Xxnds) = Coef2*txzz(Zznds, Xxnds) + Coef0*SpcSum
        txzt(:, :) = txzx(:, :) + txzz(:, :)
      END DO
    END SUBROUTINE CalWave

END MODULE WaveExtrp

!************************  TDFDEWFS2DSG  **********************
! Time Domain Finite Difference Elastic Wave Field Simulating with 2-Dimension Staggered Grid
! Written by Tche. L. from USTC, 2016.7.
! References:
!   Collino and Tsogka, 2001. Geophysics, Application of the perfectly matched absorbing layer model
     to the linear elastodynamic problem in anisotropic heterogeneous media.
!   Marcinkovich and Olsen, 2003. Journal of Geophysical Research, On the implementation of perfectly
      mathced layers in a three-dimensional fourth-order velocity-stress finite difference scheme.
!****************************************************************
PROGRAM TDFDEWFS2DSG

  USE InputPara
  USE WaveExtrp
  IMPLICIT NONE

  CHARACTER(LEN = 128) :: SnapFile = './data/Snapshot_****.dat'       ! the snapshot file name
     template.
  CHARACTER(LEN = 128) :: SyntFile = './data/SyntRcrd.dat'                       ! the synthetic
     record file name.
  REAL    :: SyntR(nrcvr, nt)
  INTEGER :: i

  CALL IntlzInputPara()
  CALL WaveExec()
  DO i = itstr,nt,ntintv
    WRITE(SnapFile(21:24),"(I4.4)") i
    CALL Output(TRIM(SnapFile), nz, nx, P(:, :, i))
  END DO
  DO i = 1,nt,1
    SyntR(:, i) = P(1, xrcvr, i)
  END DO
  CALL Output(TRIM(SyntFile), nrcvr, nt, SyntR)

END PROGRAM TDFDEWFS2DSG

SUBROUTINE Output(Outfile, M, N, OutA)
  IMPLICIT NONE
```

```
  CHARACTER(LEN = *), INTENT(IN) :: Outfile
  INTEGER, INTENT(IN) :: M, N
  REAL, INTENT(IN)     :: OutA(M, N)
  CHARACTER(LEN = 40) :: FmtStr
  INTEGER :: i, j
  INTEGER :: funit
  WRITE(FmtStr,"('(',G0,'E15.6)')") N
  OPEN(NEWUNIT = funit, FILE = Outfile, STATUS = 'UNKNOWN')
    DO i = 1,M,1
      WRITE(funit, FmtStr) (OutA(i, j), j = 1,N,1)
    END DO
  CLOSE(funit)
END SUBROUTINE Output
```

附录.3 C 程序

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#define PI 3.141592654
using namespace std;

const int nOrder = 3;
const int nTimePreSnap = 100;

typedef struct {
  int nx, nz;
  int Nx, Nz;
  int sx, sz;
  int npx, npz;
  float dx, dz;
} dim;
typedef struct {
  float *vp, *vs, *rho;
} media;
typedef struct {
  float *vxx, *vxz, *vzx, *vzz,
        *txxx, *txxz, *tzzx, *tzzz,
        *txzx, *txzz;
  float *vxt, *vzt, *txxt, *tzzt, *txzt;
} wave;
typedef struct {
  int nt;
  float dt;
  float ampl, f0, t0;
  float d0x, d0z;
  float C[nOrder];
} coeff;

void wave_exp(dim D, char *filename, float *P) {
//
  FILE *fp = fopen(filename, "wb");
    fwrite(&D.nx, sizeof(float), 1, fp);
    fwrite(&D.nz, sizeof(float), 1, fp);
    for(int i = 0; i < D.nz; i++) {
      fwrite(&P[(i + D.npz + nOrder)*D.Nx + D.npx + nOrder], sizeof(float), D.nx, fp);
    }
```

```
  fclose(fp);
/*
  FILE *fp = fopen(filename, "wt");
    for(int i = 0; i < D.nz; i++) {
      for(int j = 0; j < D.nx; j++)
        fprintf(fp, "%lf, ", (double)P[(i + D.npz + nOrder)*D.Nx + D.npx + nOrder + j]);
      fprintf(fp, "\n");
    }
  fclose(fp);
*/
}

void wave_exe(wave W, media M, dim D, coeff C) {
  int ix, iz, idx;
  int sidx = D.npx + D.sx + nOrder - 1 + (D.npz + D.sz + nOrder - 1)*D.Nx;
  float srclet;
  float *dpmlx, *dpmlz, *lambda, *mu;
  float *factor[10];

  int i;
  size_t memSize = D.Nx*D.Nz*sizeof(float);
  dpmlx  = (float*) malloc(memSize);
  dpmlz  = (float*) malloc(memSize);
  lambda = (float*) malloc(memSize);
  mu     = (float*) malloc(memSize);
  for(i = 0; i < 10; i++)
    factor[i] = (float*) malloc(memSize);

  for(idx = 0; idx < D.Nx*D.Nz; idx++) {
    mu[idx] = M.rho[idx]*M.vs[idx]*M.vs[idx];
    lambda[idx] = M.rho[idx]*M.vp[idx]*M.vp[idx] - 2*mu[idx];
  }
  for(iz = 0; iz < D.Nz; iz++)
    for(ix = 0; ix < D.Nx; ix++) {
      idx = iz*D.Nx + ix;
      if(ix < D.npx + nOrder && ix >= nOrder)
        dpmlx[idx] = C.d0x*pow(1.0*(D.npx + nOrder - ix)/D.npx, 2);
      else if(ix >= D.Nx - D.npx - nOrder && ix < D.Nx - nOrder)
        dpmlx[idx] = C.d0x*pow(1.0*(ix + D.npx + nOrder + 1 - D.Nx)/D.npx, 2);
      else
        dpmlx[idx] = 0.0;

      if(iz < D.npz + nOrder && iz >= nOrder)
        dpmlz[idx] = C.d0z*pow(1.0*(D.npz + nOrder - iz)/D.npz, 2);
      else if(iz >= D.Nz - D.npz - nOrder && iz < D.Nz - nOrder)
        dpmlz[idx] = C.d0z*pow(1.0*(iz + D.npz + nOrder + 1 - D.Nz)/D.npz, 2);
      else
        dpmlz[idx] = 0.0;
    }

  for(idx = 0; idx < D.Nx*D.Nz; idx++) {
    factor[0][idx] = (2 - C.dt*dpmlx[idx])/(2 + C.dt*dpmlx[idx]);
    factor[1][idx] = (2 - C.dt*dpmlz[idx])/(2 + C.dt*dpmlz[idx]);
    factor[2][idx] = 2*C.dt/(2 + C.dt*dpmlx[idx])/M.rho[idx]/D.dx;
    factor[3][idx] = 2*C.dt/(2 + C.dt*dpmlz[idx])/M.rho[idx]/D.dz;
    factor[4][idx] = 2*C.dt/(2 + C.dt*dpmlx[idx])*(lambda[idx] + 2*mu[idx])/D.dx;
    factor[5][idx] = 2*C.dt/(2 + C.dt*dpmlz[idx])*lambda[idx]/D.dz;
    factor[6][idx] = 2*C.dt/(2 + C.dt*dpmlx[idx])*lambda[idx]/D.dx;
    factor[7][idx] = 2*C.dt/(2 + C.dt*dpmlz[idx])*(lambda[idx] + 2*mu[idx])/D.dz;
    factor[8][idx] = 2*C.dt/(2 + C.dt*dpmlx[idx])*mu[idx]/D.dx;
    factor[9][idx] = 2*C.dt/(2 + C.dt*dpmlz[idx])*mu[idx]/D.dz;
```

```
  }

  for(idx = 0; idx < D.Nx*D.Nz; idx++) {
    W.vxx [idx] = 0.0; W.vxz [idx] = 0.0; W.vxt [idx] = 0.0;
    W.vzx [idx] = 0.0; W.vzz [idx] = 0.0; W.vzt [idx] = 0.0;
    W.txxx[idx] = 0.0; W.txxz[idx] = 0.0; W.txxt[idx] = 0.0;
    W.tzzx[idx] = 0.0; W.tzzz[idx] = 0.0; W.tzzt[idx] = 0.0;
    W.txzx[idx] = 0.0; W.txzz[idx] = 0.0; W.txzt[idx] = 0.0;
  }

  int it;
  float Psum;
  char file[200];
  for(it = 0; it < C.nt; it++) {
    if(it%nTimePreSnap == 0) printf("calculating and exporting for step %5d ...\n", it);

    for(iz = nOrder; iz < D.Nz - nOrder; iz++)
      for(ix = nOrder; ix < D.Nx - nOrder; ix++) {
        idx = iz*D.Nx + ix;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.vxt[idx + i + 1] - W.vxt[idx - i]);
        W.txxx[idx] = factor[0][idx]*W.txxx[idx] + factor[4][idx]*Psum;
        W.tzzx[idx] = factor[0][idx]*W.tzzx[idx] + factor[6][idx]*Psum;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.vzt[idx + i*D.Nx] - W.vzt[idx - (i + 1)*D.Nx]);
        W.txxz[idx] = factor[1][idx]*W.txxz[idx] + factor[5][idx]*Psum;
        W.tzzz[idx] = factor[1][idx]*W.tzzz[idx] + factor[7][idx]*Psum;
      }

    for(iz = nOrder; iz < D.Nz - nOrder; iz++)
      for(ix = nOrder; ix < D.Nx - nOrder; ix++) {
        idx = iz*D.Nx + ix;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.vzt[idx + i] - W.vzt[idx - i - 1]);
        W.txzx[idx] = factor[0][idx]*W.txzx[idx] + factor[8][idx]*Psum;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.vxt[idx + (i + 1)*D.Nx] - W.vxt[idx - i*D.Nx]);
        W.txzz[idx] = factor[1][idx]*W.txzz[idx] + factor[9][idx]*Psum;
      }
    for(idx = 0; idx < D.Nx*D.Nz; idx++)
      W.txzt[idx] = W.txzx[idx] + W.txzz[idx];

    srclet = (1 - 2*pow(PI*C.f0*((C.dt*it) - C.t0), 2))*exp(- pow(PI*C.f0*(C.dt*it - C.t0), 2));
    W.txxx[sidx] += C.ampl*srclet/4;
    W.txxz[sidx] += C.ampl*srclet/4;
    W.tzzx[sidx] += C.ampl*srclet/4;
    W.tzzz[sidx] += C.ampl*srclet/4;
    for(idx = 0; idx < D.Nx*D.Nz; idx++) {
      W.txxt[idx] = W.txxx[idx] + W.txxz[idx];
      W.tzzt[idx] = W.tzzx[idx] + W.tzzz[idx];
    }
//
    for(iz = nOrder; iz < D.Nz - nOrder; iz++)
      for(ix = nOrder; ix < D.Nx - nOrder; ix++) {
        idx = iz*D.Nx + ix;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
```

```
          Psum += C.C[i]*(W.txxt[idx + i] - W.txxt[idx - i - 1]);
        W.vxx[idx] = factor[0][idx]*W.vxx[idx] + factor[2][idx]*Psum;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.txzt[idx + i*D.Nx] - W.txzt[idx - (i + 1)*D.Nx]);
        W.vxz[idx] = factor[1][idx]*W.vxz[idx] + factor[3][idx]*Psum;
      }
    for(idx = 0; idx < D.Nx*D.Nz; idx++)
      W.vxt[idx] = W.vxx[idx] + W.vxz[idx];

    for(iz = nOrder; iz < D.Nz - nOrder; iz++)
      for(ix = nOrder; ix < D.Nx - nOrder; ix++) {
        idx = iz*D.Nx + ix;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.txzt[idx + i + 1] - W.txzt[idx - i]);
        W.vzx[idx] = factor[0][idx]*W.vzx[idx] + factor[2][idx]*Psum;
        Psum = 0.0;
        for(i = 0; i < nOrder; i++)
          Psum += C.C[i]*(W.tzzt[idx + (i + 1)*D.Nx] - W.tzzt[idx - i*D.Nx]);
        W.vzz[idx] = factor[1][idx]*W.vzz[idx] + factor[3][idx]*Psum;
      }
    for(idx = 0; idx < D.Nx*D.Nz; idx++)
      W.vzt[idx] = W.vzx[idx] + W.vzz[idx];

    if(it%nTimePreSnap == 0) {
      sprintf(file, "./data/P%05d.bin", it);
      wave_exp(D, file, W.txxt);
    }
  }

  free(dpmlx); free(dpmlz);
  free(lambda); free(mu);
  for(i = 0; i < 10; i++)
    free(factor[i]);
}

int main(int argc, char *argv[]) {

  int nx = 500, nz = 600;
  int npmlx = 20, npmlz = 20;
  int sx = 80, sz = 80;
  float dx = 5.0, dz = 5.0;
  int nt = 1000;
  float dt = 1.0e-3;
  int nppw = 12;
  float ampl = 1.0e0;

  wave W; media M; dim D; coeff C;

  int Nx, Nz;
  size_t memSize;
  int i;

  cout << "Input nt = ";
  cin >> nt;

  int prod1, prod2;
  for(int m = 1; m < nOrder + 1; m++) {
    prod1 = 1;
    for(i = 1; i <= nOrder; i++)
```

```
      if(i != m) prod1 *= (2*i - 1)*(2*i - 1);
    prod2 = 1;
    for(i = 1; i <= nOrder; i++)
      if(i != m) prod2 *= abs((2*m - 1)*(2*m - 1) - (2*i - 1)*(2*i - 1));
    C.C[m - 1] = pow(-1.0, m + 1)*prod1/(2*m - 1)/prod2;
  }

  Nx = nx + 2*npmlx + 2*nOrder;
  Nz = nz + 2*npmlz + 2*nOrder;
  memSize = Nx*Nz*sizeof(float);

  D.nx = nx; D.nz = nz;
  D.Nx = Nx; D.Nz = Nz;
  D.sx = sx; D.sz = sz;
  D.npx = npmlx; D.npz = npmlz;
  D.dx = dx; D.dz= dz;
  C.nt = nt; C.dt = dt;
  C.ampl = ampl;

  M.vp  = (float*) malloc(memSize);
  M.vs  = (float*) malloc(memSize);
  M.rho = (float*) malloc(memSize);
  for(i = 0; i < Nx*Nz; i++) {
    M.vp [i] = 2000.0;
    M.vs [i] = 1000.0;
    M.rho[i] = 1000.0;
  }

  float *vsmin = M.vs, *vsmax = M.vs;
  for(i = 1; i < Nx*Nz; i++) {
    if(*vsmin > M.vs[i]) vsmin = &M.vs[i];
    if(*vsmax < M.vs[i]) vsmax = &M.vs[i];
  }

  C.f0 = (*vsmin)/(max(dx, dz)*nppw);
  C.t0 = 1.0/C.f0;
  C.d0x = 3*(*vsmax)/dx*(8.0/15 - 3.0/100*npmlx + 1.0/1500*npmlx*npmlx);
  C.d0z = 3*(*vsmax)/dz*(8.0/15 - 3.0/100*npmlz + 1.0/1500*npmlz*npmlz);

  W.vxx  = (float*) malloc(memSize);
  W.vxz  = (float*) malloc(memSize);
  W.vzx  = (float*) malloc(memSize);
  W.vzz  = (float*) malloc(memSize);
  W.txxx = (float*) malloc(memSize);
  W.txxz = (float*) malloc(memSize);
  W.tzzx = (float*) malloc(memSize);
  W.tzzz = (float*) malloc(memSize);
  W.txzx = (float*) malloc(memSize);
  W.txzz = (float*) malloc(memSize);
  W.vxt  = (float*) malloc(memSize);
  W.vzt  = (float*) malloc(memSize);
  W.txxt = (float*) malloc(memSize);
  W.tzzt = (float*) malloc(memSize);
  W.txzt = (float*) malloc(memSize);

  wave_exe(W, M, D, C);

  free(M.vp); free(M.vs); free(M.rho);

  free(W.vxx ); free(W.vxz ); free(W.vxt );
  free(W.vzx ); free(W.vzz ); free(W.vzt );
```

```
  free(W.txxx); free(W.txxz); free(W.txxt);
  free(W.tzzx); free(W.tzzz); free(W.tzzt);
  free(W.txzx); free(W.txzz); free(W.txzt);

  return 0;
}
```

附录.4 Cuda C 程序

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#define PI 3.141592654
using namespace std;

const int nOrder = 3;
const int nTimePreSnap = 100;

typedef struct {
  int nx, nz;
  int Nx, Nz;
  int sx, sz;
  int npx, npz;
  float dx, dz;
} dim;
typedef struct {
  float *vp, *vs, *rho;
} media;
typedef struct {
  float *vxx, *vxz, *vzx, *vzz,
        *txxx, *txxz, *tzzx, *tzzz,
        *txzx, *txzz;
  float *vxt, *vzt, *txxt, *tzzt, *txzt;
} wave;
typedef struct {
  float dt;
  float d0x, d0z;
  float C[nOrder];
} coeff;
typedef struct {
  float *f0, *f1, *f2, *f3, *f4, *f5, *f6, *f7, *f8, *f9;
} factor;

__global__ void pre_eval(wave W, media M, dim D, coeff C, factor F) {
  int ix = threadIdx.x + blockIdx.x*blockDim.x;
  int iz = threadIdx.y + blockIdx.y*blockDim.y;
  int idx = iz*D.Nx + ix;
  float dpmlx = 0.0, dpmlz = 0.0;
  float lambda, mu;

  if(ix < D.Nx && iz < D.Nz) {
    mu = M.rho[idx]*M.vs[idx]*M.vs[idx];
    lambda = M.rho[idx]*M.vp[idx]*M.vp[idx] - 2*mu;
    if(ix < D.npx + nOrder && ix >= nOrder)
      dpmlx = C.d0x*pow(1.0*(D.npx + nOrder - ix)/D.npx, 2);
    if(ix >= D.Nx - D.npx - nOrder && ix < D.Nx - nOrder)
      dpmlx = C.d0x*pow(1.0*(ix + D.npx + nOrder + 1 - D.Nx)/D.npx, 2);
```

```
    if(iz < D.npz + nOrder && iz >= nOrder)
      dpmlz = C.d0z*pow(1.0*(D.npz + nOrder - iz)/D.npz, 2);
    if(iz >= D.Nz - D.npz - nOrder && iz < D.Nz - nOrder)
      dpmlz = C.d0z*pow(1.0*(iz + D.npz + nOrder + 1 - D.Nz)/D.npz, 2);

    F.f0[idx] = (2 - C.dt*dpmlx)/(2 + C.dt*dpmlx);
    F.f1[idx] = (2 - C.dt*dpmlz)/(2 + C.dt*dpmlz);
    F.f2[idx] = 2*C.dt/(2 + C.dt*dpmlx)/M.rho[idx]/D.dx;
    F.f3[idx] = 2*C.dt/(2 + C.dt*dpmlz)/M.rho[idx]/D.dz;
    F.f4[idx] = 2*C.dt/(2 + C.dt*dpmlx)*(lambda + 2*mu)/D.dx;
    F.f5[idx] = 2*C.dt/(2 + C.dt*dpmlz)*lambda/D.dz;
    F.f6[idx] = 2*C.dt/(2 + C.dt*dpmlx)*lambda/D.dx;
    F.f7[idx] = 2*C.dt/(2 + C.dt*dpmlz)*(lambda + 2*mu)/D.dz;
    F.f8[idx] = 2*C.dt/(2 + C.dt*dpmlx)*mu/D.dx;
    F.f9[idx] = 2*C.dt/(2 + C.dt*dpmlz)*mu/D.dz;

    W.vxx [idx] = 0.0; W.vxz [idx] = 0.0; W.vxt [idx] = 0.0;
    W.vzx [idx] = 0.0; W.vzz [idx] = 0.0; W.vzt [idx] = 0.0;
    W.txxx[idx] = 0.0; W.txxz[idx] = 0.0; W.txxt[idx] = 0.0;
    W.tzzx[idx] = 0.0; W.tzzz[idx] = 0.0; W.tzzt[idx] = 0.0;
    W.txzx[idx] = 0.0; W.txzz[idx] = 0.0; W.txzt[idx] = 0.0;
  }
}

__global__ void vel_eval(wave W, dim D, coeff C, factor F, int sidx) {
  int ix = threadIdx.x + blockIdx.x*blockDim.x;
  int iz = threadIdx.y + blockIdx.y*blockDim.y;
  int idx = iz*D.Nx + ix;
  int i;
  float Psum;

  if(ix >= nOrder && ix < D.Nx - nOrder && iz >= nOrder && iz < D.Nz - nOrder) {
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.txxt[idx + i] - W.txxt[idx - i - 1]);
    W.vxx[idx] = F.f0[idx]*W.vxx[idx] + F.f2[idx]*Psum;
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.txzt[idx + i*D.Nx] - W.txzt[idx - (i + 1)*D.Nx]);
    W.vxz[idx] = F.f1[idx]*W.vxz[idx] + F.f3[idx]*Psum;
    W.vxt[idx] = W.vxx[idx] + W.vxz[idx];

    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.txzt[idx + i + 1] - W.txzt[idx - i]);
    W.vzx[idx] = F.f0[idx]*W.vzx[idx] + F.f2[idx]*Psum;
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.tzzt[idx + (i + 1)*D.Nx] - W.tzzt[idx - i*D.Nx]);
    W.vzz[idx] = F.f1[idx]*W.vzz[idx] + F.f3[idx]*Psum;
    W.vzt[idx] = W.vzx[idx] + W.vzz[idx];
  }
}

__global__ void str_eval(wave W, dim D, coeff C, int sidx, float srclet, factor F) {
  int ix = threadIdx.x + blockIdx.x*blockDim.x;
  int iz = threadIdx.y + blockIdx.y*blockDim.y;
  int idx = iz*D.Nx + ix;
  int i;
  float Psum;

```

```
  if(ix >= nOrder && ix < D.Nx - nOrder && iz >= nOrder && iz < D.Nz - nOrder) {
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.vxt[idx + i + 1] - W.vxt[idx - i]);
    W.txxx[idx] = F.f0[idx]*W.txxx[idx] + F.f4[idx]*Psum;
    W.tzzx[idx] = F.f0[idx]*W.tzzx[idx] + F.f6[idx]*Psum;
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.vzt[idx + i*D.Nx] - W.vzt[idx - (i + 1)*D.Nx]);
    W.txxz[idx] = F.f1[idx]*W.txxz[idx] + F.f5[idx]*Psum;
    W.tzzz[idx] = F.f1[idx]*W.tzzz[idx] + F.f7[idx]*Psum;

    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.vzt[idx + i] - W.vzt[idx - i - 1]);
    W.txzx[idx] = F.f0[idx]*W.txzx[idx] + F.f8[idx]*Psum;
    Psum = 0.0;
    for(i = 0; i < nOrder; i++)
      Psum += C.C[i]*(W.vxt[idx + (i + 1)*D.Nx] - W.vxt[idx - i*D.Nx]);
    W.txzz[idx] = F.f1[idx]*W.txzz[idx] + F.f9[idx]*Psum;
    W.txzt[idx] = W.txzx[idx] + W.txzz[idx];

    if(idx == sidx) {
      W.txxx[idx] += srclet/4;
      W.txxz[idx] += srclet/4;
      W.tzzx[idx] += srclet/4;
      W.tzzz[idx] += srclet/4;
    }
    W.txxt[idx] = W.txxx[idx] + W.txxz[idx];
    W.tzzt[idx] = W.tzzx[idx] + W.tzzz[idx];
  }
}

void exp_wave(dim D, char *filename, float *P) {
//
  FILE *fp = fopen(filename, "wb");
    fwrite(&D.nx, sizeof(float), 1, fp);
    fwrite(&D.nz, sizeof(float), 1, fp);
    for(int i = 0; i < D.nz; i++) {
      fwrite(&P[(i + D.npz + nOrder)*D.Nx + D.npx + nOrder], sizeof(float), D.nx, fp);
    }
  fclose(fp);
/*
  FILE *fp = fopen(filename, "wt");
    for(int i = 0; i < D.nz; i++) {
      for(int j = 0; j < D.nx; j++)
        fprintf(fp, "%lf, ", (double)P[(i + D.npz + nOrder)*D.Nx + D.npx + nOrder + j]);
      fprintf(fp, "\n");
    }
  fclose(fp);
*/
}

int main(int argc, char *argv[]) {

  int nx = 500, nz = 600;
  int npmlx = 20, npmlz = 20;
  int sx = 80, sz = 80;
  float dx = 5.0, dz = 5.0;
  int nt = 1000;
  float dt = 1.0e-3;
```

```
int nppw = 12;
float ampl = 1.0e0;

wave W; media M; dim D; coeff C; factor F;

int Nx, Nz;
size_t memSize;
int i, j;

cout << "Input nt = ";
cin >> nt;

int prod1, prod2;
for(int m = 1; m < nOrder + 1; m++) {
  prod1 = 1;
  for(i = 1; i <= nOrder; i++)
    if(i != m) prod1 *= (2*i - 1)*(2*i - 1);
  prod2 = 1;
  for(i = 1; i <= nOrder; i++)
    if(i != m) prod2 *= abs((2*m - 1)*(2*m - 1) - (2*i - 1)*(2*i - 1));
  C.C[m - 1] = pow(-1.0, m + 1)*prod1/(2*m - 1)/prod2;
}

Nx = nx + 2*npmlx + 2*nOrder;
Nz = nz + 2*npmlz + 2*nOrder;
memSize = Nx*Nz*sizeof(float);

D.nx = nx; D.nz = nz;
D.Nx = Nx; D.Nz = Nz;
D.sx = sx; D.sz = sz;
D.npx = npmlx; D.npz = npmlz;
D.dx = dx; D.dz= dz;

float *Vp  = (float*) malloc(memSize);
float *Vs  = (float*) malloc(memSize);
float *Rho = (float*) malloc(memSize);
for(i = 0; i < Nx*Nz; i++) {
  Vp [i] = 2000.0;
  Vs [i] = 1000.0;
  Rho[i] = 1000.0;
}

cudaMalloc((float**) &M.vp , memSize);
cudaMalloc((float**) &M.vs , memSize);
cudaMalloc((float**) &M.rho, memSize);
cudaMemcpy(M.vp , Vp , memSize, cudaMemcpyHostToDevice);
cudaMemcpy(M.vs , Vs , memSize, cudaMemcpyHostToDevice);
cudaMemcpy(M.rho, Rho, memSize, cudaMemcpyHostToDevice);

float *vsmin = Vs, *vsmax = Vs;
for(i = 1; i < Nx*Nz; i++) {
  if(*vsmin > Vs[i]) vsmin = &Vs[i];
  if(*vsmax < Vs[i]) vsmax = &Vs[i];
}

float f0, t0;
f0 = (*vsmin)/(max(dx, dz)*nppw);
t0 = 1.0/f0;
C.dt = dt;
C.d0x = 3*(*vsmax)/dx*(8.0/15 - 3.0/100*npmlx + 1.0/1500*npmlx*npmlx);
C.d0z = 3*(*vsmax)/dz*(8.0/15 - 3.0/100*npmlz + 1.0/1500*npmlz*npmlz);
```

```
cudaMalloc((float∗∗) &F.f0, memSize);
cudaMalloc((float∗∗) &F.f1, memSize);
cudaMalloc((float∗∗) &F.f2, memSize);
cudaMalloc((float∗∗) &F.f3, memSize);
cudaMalloc((float∗∗) &F.f4, memSize);
cudaMalloc((float∗∗) &F.f5, memSize);
cudaMalloc((float∗∗) &F.f6, memSize);
cudaMalloc((float∗∗) &F.f7, memSize);
cudaMalloc((float∗∗) &F.f8, memSize);
cudaMalloc((float∗∗) &F.f9, memSize);

cudaMalloc((float∗∗) &W.vxx , memSize);
cudaMalloc((float∗∗) &W.vxz , memSize);
cudaMalloc((float∗∗) &W.vzx , memSize);
cudaMalloc((float∗∗) &W.vzz , memSize);
cudaMalloc((float∗∗) &W.txxx, memSize);
cudaMalloc((float∗∗) &W.txxz, memSize);
cudaMalloc((float∗∗) &W.tzzx, memSize);
cudaMalloc((float∗∗) &W.tzzz, memSize);
cudaMalloc((float∗∗) &W.txzx, memSize);
cudaMalloc((float∗∗) &W.txzz, memSize);
cudaMalloc((float∗∗) &W.vxt , memSize);
cudaMalloc((float∗∗) &W.vzt , memSize);
cudaMalloc((float∗∗) &W.txxt, memSize);
cudaMalloc((float∗∗) &W.tzzt, memSize);
cudaMalloc((float∗∗) &W.txzt, memSize);

dim3 Block(32, 16);
dim3 Grid(ceil(1.0∗Nx/Block.x), ceil(1.0∗Nz/Block.y));

cout << ”Block = ” << Block.x << ” ” << Block.y << endl;
cout << ”Grid = ” << Grid.x << ” ” << Grid.y << endl;

float ∗P;
P = (float∗) malloc(memSize);

float srclet;
int sidx = npmlx + sx + nOrder − 1 + (npmlz + sz + nOrder − 1)∗D.Nx;
pre_eval <<< Grid, Block >>> (W, M, D, C, F);

int it = 0;
char file[200];
for(i = 0; i∗nTimePreSnap < nt; i++) {
  printf(”calculating and exporting for step %5d ...\n”, it);
  for(j = 0; j < nTimePreSnap; j++, it++) {
    if(it > nt) break;
    srclet = ampl∗(1 − 2∗pow(PI∗f0∗(dt∗it − t0), 2))∗exp( − pow(PI∗f0∗(dt∗it − t0), 2));

    str_eval <<< Grid, Block >>> (W, D, C, sidx, srclet, F);
    vel_eval <<< Grid, Block >>> (W, D, C, F, sidx);

    if((it − 1)%nTimePreSnap == 0) {
      cudaDeviceSynchronize();
      sprintf(file, ”./data/P%05d.bin”, it − 1);
      cudaMemcpy(P, W.txxt, memSize, cudaMemcpyDeviceToHost);
      exp_wave(D, file, P);
    }
  }
}
```

```
  free(Vp); free(Vs); free(Rho);
  cudaFree(M.vp); cudaFree(M.vs); cudaFree(M.rho);

  free(P);

  cudaFree(W.vxx ); cudaFree(W.vxz ); cudaFree(W.vxt );
  cudaFree(W.vzx ); cudaFree(W.vzz ); cudaFree(W.vzt );
  cudaFree(W.txxx); cudaFree(W.txxz); cudaFree(W.txxt);
  cudaFree(W.tzzx); cudaFree(W.tzzz); cudaFree(W.tzzt);
  cudaFree(W.txzx); cudaFree(W.txzz); cudaFree(W.txzt);

  return 0;
}
```