Environmental Modelling & Software 52 (2014) 207–221

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

Vectorized simulation of groundwater flow and streamline transport

George Kourakos, Thomas Harter*

Dept. of Land Air and Water Resources, University of California Davis, Davis, 95616 CA, USA

ARTICLE INFO

Article history:
Received 11 April 2013
Received in revised form
21 October 2013
Accepted 26 October 2013
Available online 27 November 2013

Keywords:
Non-point source pollution
Finite element method
Matlab
Streamline transport
Groundwater modeling

ABSTRACT

We describe a modeling suite of Matlab functions for simulating nonpoint source (NPS) pollution in groundwater aquifers. The NPS model simulates groundwater flow and contaminant transport from a large array (order of $10^2 - 10^7$) of spatially distributed sources with time-varying pollution strength to a similarly large array of spatially distributed production wells (receptors) using the streamline transport approach. The code solves three equations: steady-state groundwater flow, particle tracking, and transient advection dispersion contaminant transport. The code performs convolution integration in its predictive step. Written in highly efficient vectorized form to avoid time consuming "for/while" loops, the code is also suitable for other groundwater flow and transport problems. The code is verified against analytical solutions and finite element software Comsol. An application illustrates 200 years of transient nitrate transport in the 2000 km^2 Tule River aquifer sub-basin of the Central Valley, California, with 9000 individual nitrate sources and 1920 wells.

Software availability

Name of Software: mSim
Developers: George Kourakos, Thomas Harter
First available year: 2013
Program Language: Matlab, C/C++
Availability: http://groundwater.ucdavis.edu/mSim/
Primary contacts: George Kourakos, Thomas Harter
E-mail: gkourakos@ucdavis.edu, giorgk@gmail.com, thharter@ucdavis.edu

1. Introduction

Numerical simulation models have been established as standard methods for studying a vast variety of physical phenomena and environmental processes including water resources (Moriasi et al., 2012; Bobba, 2012; Bennett et al., 2013). In many countries, water management decisions are based, to a large extent, on simulation with numerical models (Refsgaard and Henriksen, 2004). Hence, numerical models have found broad applicability in research and education as well as in consulting, decision making, and industry. Following this trend, groundwater hydrologists also have a long history of developing and using numerical models for simulating groundwater flow and contaminant transport i.e. Modflow (Harbaugh et al., 2000), Parflow (Ashby and Falgout, 1996), HydroGeoSphere (Therrien et al., 2007), Feflow (Trefry and Muffels, 2007), SUTRA (Voss, 1984), HYDRUS (Šimunek et al., 2012), IWFM (Integrated Water Flow Model, 2012), FEMWATER (Lin et al., 1997) etc.

In general, numerical models can be divided into two broad categories: Open source, where the code is available to the user under certain license types (e.g. Modflow, IWFM, FEMWATER) and closed source (e.g. HydroGeoSphere, Feflow, Comsol (COMSOL, 2008)) where the user has no access to the original code. Typically, closed source software products are commercial products supported through license sales (Feflow, Comsol). Neither type is necessarily preferable. The choice depends on user experience and needs and on budget constraints. Closed source models, especially those available commercially, are typically accompanied by user-friendly graphical user interfaces, which attempt to assist and protect the user from dealing with many of the details involved in a numerical simulation, i.e. grid generation, assembly process, preconditioners, solvers, parallel implementation, etc. While this is desirable in many cases, it is also very important, especially in research, that users have access and be able to deepen and intervene with the numerical codes. In addition, open source codes allow widespread reproduction of results, which is fundamental to science and useful to a wider audience than closed source codes. Hence, open source codes are more common in education and research, while commercial products are more popular in consulting and industry. Open source codes, while transparent to the user, are not always user friendly. For example,

* Corresponding author.
E-mail addresses: gkourakos@ucdavis.edu, giorgk@gmail.com (G. Kourakos), thharter@ucdavis.edu (T. Harter).

1364-8152/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.envsoft.2013.10.029

open source codes such as MODFLOW and FEMWATER, which simulate the groundwater flow equation and saturated/unsaturated, density driven, flow and transport respectively, require a tedious input data preparation process, even for relatively simple applications. For the aforementioned codes, pre- and post-processors have been developed that assist in the data input and output processing, e.g. GroundwaterVista™, Visual Modflow™, GMS™, ModelMuse (Winston, 2009), etc.

The majority of open source codes is written in FORTRAN (Modflow, IWFM, Hydrogeosphere, SUTRA, FEMWATER) or C/C++ (Comsol, OpenFoam (OpenFoam, 2013), Dune (Dedner et al., 2011)), which provide strong computational efficiency. However reading and understanding languages such as FORTRAN and C/C++ requires significant experience and knowledge of the programming language itself. Implementing and testing research ideas that require modifications to an existing code are potentially difficult and error prone tasks. In addition, most recent codes are likely to use an object oriented programming style e.g. C++, which is convenient for the experienced developer, yet adds another level of difficulty to users that are not programmers.

In addition to products based on high performance languages such as Fortran and C/C++, there is a number of high-level scripting languages that promise to increase productivity without compromising performance on high performance computing systems (Chaves et al., 2006) such as Python, Matlab, Octave etc. In addition these languages are thought to have less steep learning curves when compared to Fortran or C/C++. In particular, Matlab is a proprietary platform that is very popular among engineering and environmental sciences students. It provides an interactive environment for numerical computation, visualization, and programming, yet the code is visible to the user and available for further modifications. Octave on the other hand is an open source programming language under the GNU license, which is quite similar to Matlab, so that most programs are easily portable to Matlab. But Octave currently lacks an interactive and user friendly debugging environment.

Matlab and Octave codes are not considered as efficient as codes written in compiled programming languages, yet there is a growing number of examples where Matlab is used to carry out computationally intensive tasks, such as mesh generation (Persson and Strang, 2004; Talischi et al., 2012), model emulation (Tych and Young, 2012), remote sensing (Teza et al., 2012), numerical simulations (Lee et al., 2004; Kattan, 2008; Louwyck et al., 2012; Lie et al., 2012; Kumar and Dodagoudar, 2010), optimization (Kourakos and Mantoglou, 2008; Kourakos and Mantoglou, 2012a), neural networks (Kourakos and Mantoglou, 2012b), parallel computing (Kepner, 2009) etc. The major drawback of Matlab and Octave is that the execution of repeated tasks in the form of *for* and *while* loop becomes very time consuming when compared to compiled languages. To alleviate this shortcoming, Matlab and Octave provide a "vectorized" formulation of loops. According to Matlab's definition, vectorization is the conversion of "for" and "while" loops to equivalent vector or matrix operations which can be performed in Matlab at a speed comparable to FORTRAN and C/C++. The efficiency of vectorized implementation for assembling system matrices that arise from the numerical solution of partial differential equation has been reported by various studies (Higham, 2002; Koko, 2007; Funken et al., 2011; Andreassen et al., 2011), while Aslam and Hendren (2012) developed a framework to optimize Matlab code.

In this paper we develop a Matlab program, which we will refer to as mSim. The code is intended for the simulation of Non-Point Source (NPS) pollution in groundwater aquifers. The majority of the Matlab code is portable to Octave without any modification. Yet there is a small number of preprocessing functions, which are used to convert the geometrical description of a study area to a constructive solid geometry object, that are written in the new style of Matlab classes introduced in the 2008a edition, which is currently not supported by Octave.

Generally, tools applied to non-point source assessment can be grouped into three categories (NRC, 1993): 1) Index-based methods which combine spatial properties of the study area such as soil type, slope, climate, depth to groundwater, etc. and provide vulnerability maps (Aller et al., 1987); 2) statistical methods such as regression models (Nolan and Hitt, 2006), neural networks (Al-Mahallawi et al., 2012), etc. 3) physically based methods attempt to simulate the fate of contaminants in groundwater by solving the flow and transport equations. Due to the large extent of NPS pollution in groundwater basins, full 3D transport simulation is limited by the available computational resources to simplified cases (Gallardo et al., 2005; Jiang and Somers, 2009). A promising alternative to full 3D transport simulation is the application of streamline transport methods (Fogg et al., 1999; McMahon et al., 2008; Starn et al., 2012).

To simulate NPS we utilize the streamline modeling framework proposed by Kourakos et al. (2012) (NPSAT – NonPoint Source Assessment Toolbox). The NPSAT consists of three major processes: i) simulation of steady state groundwater flow, ii) backward particle tracking, and iii) 1D transient transport simulation along the streamlines. Note that the governing equations used in the NPSAT also describe a variety of environmental processes, therefore the code can be used for simulations other than NPS modeling that are governed by the same equations.

In our implementation we take advantage of the capabilities of Matlab and develop a highly vectorized code, minimizing the number of loops as much as possible. Hence we are able to solve problems up to several million degrees of freedom, while keeping the assembly and solution processes on the order of minutes. Matlab/Octave-provided solvers are only used for the solution of relatively small sparse systems. For large systems, a variety of existing solvers can seamlessly be combined with mSim, for example, pyAMG (Bell et al., 2011), which is a python implementation of the Algebraic Multigrid solvers, and HYPRE (HYPRE, 2012) and Trilinos (Heroux et al., 2003), which provide rich libraries for solving large sparse linear systems of equations on massively parallel computers.

This paper is divided into six sections. The second section presents an overview of mSim. The third section briefly describes the governing equations employed by NPSAT and its finite element formulation. The fourth section discusses the vectorized implementation. We then describe a validation of mSim against analytical and numerical codes and apply mSim to a real case study in south-central California, USA. The last section summarizes the key points of this study.

2. Overview of mSim

The mSim code is a collection of Matlab functions that are primarily intended for the simulation of non-point source (NPS) pollution in agricultural groundwater basins. The modeling approach, assumptions and justifications are described in detailed in Kourakos et al. (2012). In this paper we focus on efficient code development based on the high-level language Matlab/Octave. The processes that are involved with the NPS pollution modeling are the simulation of groundwater flow, of particle tracking and of solute transport along particle streamlines based on the solution of the advection–dispersion equation.

The Matlab/Octave functions are organized into five groups: i) mFlow, ii) mPart, iii) mTrans, iv) mUtil and v) mNPSAT, where each group is actually a directory that contains the required files for the simulation of flow, the particle tracking, the simulation of transport,