REAL-TIME MODELING OF RIVER BASIN RESPONSE USING RADAR-GENERATED RAINFALL MAPS AND A DISTRIBUTED HYDROLOGIC DATABASE

by
Luis Garrote

Ingeniero de Caminos, Canales y Puertos
Universidad Politécnica de Madrid, 1985

Doctor Ingeniero de Caminos, Canales y Puertos
Universidad Politécnica de Madrid, 1990

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Civil Engineering

and

Civil Engineer

at the
Massachusetts Institute of Technology
August, 1992

Signature of Author... .....
Department of Civil and Environmental Engineering
August 12, 1992

Certified by.....
Professor Rafael L. Bras
Thesis Supervisor

Accepted by.....
Eduardo Kausel
Chairman, Departmental Committee on Graduate Studies

Real-Time Modeling of River Basin Response Using Radar-Generated Rainfall Maps and a Distributed Hydrologic Database

by
Luis Garrote

Submitted to the Department of Civil and Environmental Engineering on August 12, 1992 in partial fulfillment of the requirements for the degrees of Master of Science in Civil Engineering and Civil Engineer.

ABSTRACT

A distributed model for real-time rainfall-runoff simulation during floods is presented. The model is called Distributed Basin Simulator. DBS uses information from a distributed hydrologic database to interpret rainfall information in real time. The model is largely based on the detailed topographical information provided by digital elevation models (DEM). Basin representation adopts the rectangular grid of the DEM, and other soil properties, input data and state variables are also represented as data layers using the same scheme. The basic objective is to map the topographically-driven evolution of saturated areas as the storm progresses. Two modes of runoff generation are simulated: infiltration excess runoff and return flow. DBS applies a kinematic model of infiltration to evaluate local runoff generation in grid elements, and also accounts for lateral moisture flow between elements in a simplified manner. The model was successfully calibrated for the Sieve basin.

Object-oriented methodologies were applied in model design and implementation. The resulting computer package is called Real-time Interactive Basin Simulator. RIBS combines the distributed basin simulator and a hydrologic database within an interactive real-time framework. Model structure is flexible, and several modes of operation are possible: an off-line calibration mode, an on-line simulation mode and an on-line forecasting mode. RIBS has two graphic user interfaces: a synchronous, model-driven interface and an asynchronous, user-driven interface. The synchronous interface displays simulation results and forecasts in real time as the model progresses. The asynchronous user interface offers off-line information of basin state and model results at the user's request. It can generate hydrographs at any point within the basin, display the time evolution of model variables for any grid element, or represent the spatial distribution of basic or derived variables.

Thesis Supervisor: Dr. Rafael L. Bras
Title: Leonhard Professor of Civil Engineering

Acknowledgements

Support for my stay in MIT has been provided by the Spanish Ministry of Education and Science, through its program of Education of University-Level Faculty. I am deeply grateful to all Spanish taxpayers who, unknowingly, added this load to their already overburdened income deductions and offered me the opportunity to conduct this research in MIT. The research was also supported by the U.S. Army Research Office (Grant DAALO-3-89-K-0151), the Arno Project of the National Research Council of Italy, the National Weather Service (cooperative agreement NA86AA-D-HY123) and the National Science Foundation (Grant CES-8815725). With the joint sponsorship of three countries, I can only wish that this work was worth it.

I also wish to thank Rafael Bras, my advisor, who combined direction, scientific advice and resources with a great deal of patience. I imagine that his profound knowledge of stochastic processes helped him understand my somewhat random walk through this deterministic field. Rafael, Ignacio Rodríguez-Iturbe, Dara Entekabi and other members of MIT faculty never ceased to amaze me for the breadth of their knowledge and their capacity to arise my interest in so many and so different topics.

This work is the result of the collective efforts of many individuals. Our research in digital elevation models was initiated by David Tarboton, who opened more lines of investigation than we can possibly cover. Mariza Cabral developed the infiltration model used here and built a first prototype of the distributed model. Fabio Castelli contributed significantly during the

initial stages of software design and wrote the original version of the rainfall-acquisition module and several programs to deal with raster files. The work has greatly benefited from comments and suggestions by Marcos Pessoa, who managed to stay calm and conduct fruitful research despite the unnerving imperfections of the early versions of the model. All these contributions are gratefully acknowledged.

Other members of our research group, José Reyes, Shafiqul Islam, Fatih Eltahir, Ede Ijjasz, Glenn Moglen, Qihang Li and Jinfeng Wang frequently made themselves available for discussion, and provided extremely helpful comments during our group meetings. I also extend the acknowledgement to all the friends I have made in the Parsons Lab, who helped me switch my interest from the easy pleasures of life to the more rewarding field of the activities of the intellect.

Table of Contents

Abstract	3
Acknowledgements	5
Table of contents	7
List of figures	11
List of tables	19
1 Introduction	**21**
1.1 Introduction, motivation and scope of the work	21
1.2 Previous work on distributed modeling	26
1.3 Outline of approach	36
2 The Rainfall-Runoff Transformation Model	**43**
2.1 Basin representation	43
2.2 Runoff generation	48
2.2.1 The one-dimensional model of infiltration	49
2.2.2 Adaptation to basin scale	63
2.2.3 Moisture balance	65
2.2.4 Runoff generation	74
2.3 Surface flow routing	81
2.3.1 Basic routing equation	82
2.3.2 Distributed instantaneous response function	84
2.3.3 Non-linearities in basin response	85
3 Model Performance	**91**
3.1 Sensitivity analyses	91
3.1.1 Hillslope model	93
3.1.2 Basin model	109
3.2 Model calibration and evaluation	154
3.2.1 Calibration methodology	154
3.2.2 Results of the calibration process	162
3.2.3 Results of the evaluation step	167

4 Real-Time Use of the Model 175
4.1 Real-time operation of physically-based models during floods 175
4.1.1 Scope of the problem 176
4.1.2 Requirements of real-time flood forecasting system 178
4.2 System concept 184
4.2.1 Design considerations 185
4.2.2 System architecture 187
4.2.3 Process handling 190
4.2.4 Data handling 195
4.2.5 General management 201
4.3 RIBS Prototype 206
4.3.1 Modeling objectives 207
4.3.2 Architecture 209
4.3.3 Modes of operation 214
4.4 Functional units 218
4.4.1 Rainfall-runoff functional units 219
4.4.2 User interface functional units 224

5 Software Design for the Distributed Basin Simulator 229
5.1 Design overview 229
5.1.1 Programming methodology 230
5.1.2 The *Pixel* object 235
5.1.3 The *Basin* object 238
5.1.4 The *Gauge* object 241
5.1.5 The *Simulator* object 244
5.2 Implementation of the 1-D model of infiltration 246
5.2.1 Methods of the Pixel object 246
5.2.2 The one-dimensional model of infiltration 248
5.3 Implementation of basin response 258
5.3.1 Implementation of runoff generation 259
5.3.2 Implementation of surface flow routing 263
5.4 Implementation of simulation management 265
5.4.1 Input-output operations 266
5.4.2 Model operation 267

6 Software Design for the User Interface 271
6.1 Design overview 271
6.1.1 The X Windows programming environment 272
6.1.2 The concept of RIBS user interface 274
6.2 The model-driven user interface 279
6.2.1 Basic objects 281
6.2.2 RIBS graphic objects 284
6.2.3 Executable models 290
6.2.4 The model-driven interface 293
6.3 The user-driven interface 294
6.3.1 The *Viewer* object 296
6.3.2 The executable module 303

7 Conclusion 307
7.1 Summary of conclusions 307
7.2 Future work 310

References 315

Appendix 1
A Kinematic Model of Infiltration for Vertically Heterogeneous, Anisotropic and Sloped Soils 325

Appendix 2
Data Management 369

Appendix 3
RIBS User Manual 389

Appendix 4
RIBS Sample Run 433

Appendix 5
Software Documentation 447

List of Figures

Figure		Page
2.1	Representation of distributed information in the basin. All data layers share the same rectangular grid.	44
2.2	Model representation of a subgrid element.	46
2.3	Schematic representation of the wetting and the top fronts The horizontal axis represents moisture content θ, and the vertical axis represents normal depth.	53
2.4	Schematic representation of unsaturated flow.	55
2.5	Schematic representation of saturated flow. Positive pressure corresponds to the equivalent hydraulic conductivity of the saturated area as a whole.	59
2.6	Representation of the four possible runoff-generation states of the pixel.	75
3.1	Schematic representation of the geometry of the hillslope used in the sensitivity analysis.	94
3.2	Hillslope models used in the sensitivity analysis.	95
3.3	Terrain surface and front positions with respect to the water table level after 30 and 60 hours of simulation. The results correspond to hillslope model a. The pixel size is 1 m and the computation time step is 5 min.	97
3.4	Sensitivity of front position to pixel size for computation time steps of 5, 30 and 60 min after 30 hours of simulation. The results correspond to hillslope model a.	98
3.5	Sensitivity of front position to pixel size for computation time steps of 5, 30 and 60 min after 60 hours of simulation. The results correspond to hillslope model a.	99
3.6	Sensitivity of front position to computation time step for pixel size of 1, 10 and 100 m after 30 hours of simulation. The results correspond to hillslope model a.	101

3.7 Sensitivity of front position to computation time step for pixel size of 1, 10 and 100 m after 60 hours of simulation. The results correspond to hillslope model a. 102

3.8 Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 1. 105

3.9 Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 10. 106

3.10 Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 100. 107

3.11 Boundaries and topography of the Sieve river basin. 110

3.12a Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 10% probability of exceedance and the parameter R_i interpreted as initial moisture content. 114

3.12b Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 10% probability of exceedance and the parameter R_i interpreted as initial recharge rate. 115

3.13a Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 50% probability of exceedance and the parameter R_i interpreted as initial moisture content. 116

3.13b Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 50% probability of exceedance and the parameter R_i interpreted as initial recharge rate. 117

3.14a Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 90% probability of exceedance and the parameter R_i interpreted as initial moisture content. 118

3.14b Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 90% probability of exceedance and the parameter R_i interpreted as initial recharge rate. 119

3.15a Time evolution of different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content. 121

3.15b Time evolution of different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate. 122

3.16a Decomposition of basin response into different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content. 123

3.16b Decomposition of basin response into different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate. 124

3.17a Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 10% probability of exceedance and parameter R_i interpreted as initial moisture content. 127

3.17b Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 10% probability of exceedance and parameter R_i interpreted as initial recharge rate. 128

3.18a Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content. 129

3.18b Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate. 130

3.19a Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 90% probability of exceedance and parameter R_i interpreted as initial moisture content. 131

3.19b Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 90% probability of exceedance and parameter R_i interpreted as initial recharge rate. 132

3.20a Time evolution of different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content. 134

3.20b Time evolution of different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate. 135

3.21a Decomposition of basin response into different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content. 136

3.21b Decomposition of basin response into different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate. 137

3.22a Sensitivity of basin response to initial state for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content. 139

3.22b Sensitivity of basin response to initial state for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate. 140

3.23a Sensitivity of basin response to initial state for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and parameter R_i interpreted as initial moisture content. 141

3.23b Sensitivity of basin response to initial state for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and parameter R_i interpreted as initial recharge rate. 142

3.24a Time evolution of different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content. 144

3.24b Time evolution of different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate. 145

3.25a Decomposition of basin response into different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content. 146

3.25b Decomposition of basin response into different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate. 147

3.26 Sensitivity of basin response to the interpretation of parameter R_i for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and initial state with 50% probability of exceedance. 148

3.27 Sensitivity of basin response to the interpretation of parameter R_i for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and initial state with 50% probability of exceedance. 149

3.28 Sensitivity of basin response to the interpretation of parameter R_i for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and anisotropy ratio equal to 10. 149

3.29 Time evolution of different modes of runoff generation for interpretation of parameter R_i as initial moisture content and as initial recharge rate. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and initial state with 50% probability of exceedance. 150

3.30 Decomposition of basin response into different modes of runoff generation for interpretation of parameter R_i as initial moisture content and as initial recharge rate. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and initial state with 50% probability of exceedance. 150

3.31 Sensitivity of basin response to the ratio of stream and hillslope velocities K_v. The results correspond to coefficient c_v equal to 400 mh^{-1} and exponent r equal to 0.1. 152

3.32 Sensitivity of basin response to coefficient c_v. The results correspond to velocity ratio K_v equal to 10 and exponent r equal to 0.1. 153

3.33 Sensitivity of basin response to exponent r. The results correspond to velocity ratio K_v equal to 10 and coefficient c_v equal to 400 mh^{-1}. 153

3.34 Antecedent rainfall corresponding to the storms in the calibration set. The plot shows cummulative rainfall in mm preceeding the storm as a function of time in days. 159

3.35 Observed and simulated hydrographs for the storm of February 1977. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance. 163

3.36 Observed and simulated hydrographs for the storm of February 1979. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance. 164

3.37	Observed and simulated hydrographs for the storm of November 1982. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	164
3.38	Observed and simulated hydrographs for the storm of February 1983. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	165
3.39	Observed and simulated hydrographs for the storm of January 1985. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	165
3.40	Observed and simulated hydrographs for the storm of December 1968. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	168
3.41	Observed and simulated hydrographs for the storm of January 1969. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	169
3.42	Observed and simulated hydrographs for the storm of December 1975. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	169
3.43	Observed and simulated hydrographs for the storm of December 1976. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	170
3.44	Observed and simulated hydrographs for the storm of November 1987. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	170
3.45	Observed and simulated hydrographs for the storm of November 1991. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.	172
4.1	Schematic representation of the architecture of *RIBS*	188
4.2	Processes in a real-time flood-forecasting system	191
4.3	Data flow in a real-time flood-forecasting system	198
4.4	Process dependencies for the actual modules of *RIBS*	213
4.5	Organization of hydrologic functional units	220
4.6	Example of a pixel display	225
4.7	Example of a hydrograph display	226

4.8	Example of a basin display	227
5.1	High-level components of the distributed basin simulator	234
5.2	Structure of the *Pixel* object	236
5.3	Structure of the *Basin* object	239
5.4	Structure of the *Gauge* object	242
5.5	Structure of the *Simulator* object	244
6.1	High-level components of the model-driven user interface	280
6.2	Structure of the basic graphic objects	281
6.3	Structure of the *Image* class	285
6.4	Basic window layout for the object viewers	291
6.5	Structure of the *Viewer* object	297
6.6	Basic window layout for the user-driven interface	302
6.7	Options in the pulldown menus of the basin viewer	303

List of Tables

Table		Page
3.1	Soil characteristics of the hillslope models	94
3.2	Model parameters for the hillslope models	96
3.3	Model parameters for the basin sensitivity analyses	111
3.4	Parameter values for the sensitivity to surface routing	151
3.5	Storms used in the calibration step	155
3.6	Original data in the soil study and values adopted in the model (from Cabral et al., 1990)	157
3.7	Cumulative antecedent precipitation (mm) for the storms in the calibration set	160
3.8	Parameter set which gave the best fit in calibration	162
3.9	Calibration. Summary of results	166
3.10	Storms used in the evaluation step	168
3.11	Evaluation. Summary of results	171

CHAPTER 1

Introduction

1.1 Introduction, motivation and scope of the work

This work is concerned with the practical application of distributed modeling to real-time flood forecasting. Most distributed models reported in the literature have been applied at scales smaller than those generally of interest to the operational hydrologist. The emphasis is usually placed on the scientific understanding of the physical processes governing runoff generation at the hillslope scale. Extrapolation of those basic modeling strategies to midsize or large basins (hundreds of km^2) leads either to serious problems of representation of subgrid variability or to unfeasible data and computational requirements. However, active research addressing both issues is currently in progress. Certainly, developments in computer technology have outpaced progress in catchment hydrology, but many authors remain optimistic about the future of distributed modeling. It is therefore reasonable to expect that in the near future the results of research on distributed modeling will be made available to consulting engineers in the form of software packages including distributed rainfall-runoff models (Beven, 1989)

The application of distributed modeling schemes in real time is the focus of this work. Two aspects make distributed modeling specially attractive for flood forecasting: the possibility to include detailed

topographical information and the opportunity to make effective use of radar-generated rainfall maps. Flood forecasting schemes based on rainfall-runoff modeling are usually applied to midsize catchments, where storm size is comparable to basin size, and basin relief is relatively easy to characterize.

Aerial photogrametry and computer information processing in the form of digitized elevation maps have made topographical information widely available at a relatively low cost. Obtaining other types of hydrological information, such as vegetative cover, geology, soil types, etc., is comparatively more expensive, since it must rely on less-developed remote sensing techniques or intensive field work. Topography has been proven to affect runoff generation dramatically, specially in humid climates. The combined effects of a dense vegetative cover and moisture availability facilitate the development of highly permeable soil horizons. In these basins, runoff generation is mainly controlled through hillslope processes, which determine the expansion of saturated areas in troughs and depressions. In the absence of long historic rainfall-runoff records to calibrate simpler models, physically-based modeling is the only viable option to incorporate available information into the modeling process. Hence, topography-based models are a valid alternative.

The availability of radar-generated rainfall maps is also a strong argument in favor of distributed modeling. Distributed models are an excellent tool to track the spatial and temporal evolution of atmospheric disturbances and to evaluate their effects on basin response. The spatial structure of rainfall, as identified in radar pictures, is very complex and highly variable. High-intensity cells develop against a background of considerably less intense rainfall. In these conditions, the highly non-

linear character of runoff generation raises questions about the validity of lumped schemes, based on spatial averages of rainfall. If distributed rainfall information is available, a modeling scheme that accounts for the spatial distribution of rainfall on a spatially heterogeneous basin can, in principle, outperform lumped schemes.

Remotely-sensed rainfall should also be included in operational flood forecasting because short-term rainfall forecasting methods can greatly benefit from the characterization of a storm as a spatial phenomenon (Collinge and Kirkby, 1987). Detailed descriptions of storm development can be used to identify mesoscale weather phenomena and to provide forecasts based on linear extrapolations for short periods of time (Campbell and Olson, 1986; Seo and Smith, 1992). In this context, complex situations in which a storm moves from one basin into another or where only a fraction of the basin is covered by the storm can be effectively analyzed with a distributed modeling scheme.

This work is therefore based on the conviction that a physically-based model using detailed topographical information combined with a less precise soil characterization is the most cost-effective strategy currently available to address the problem of flood forecasting in ungaged watersheds. There are, of course, the challenges of developing accurate quantitative real-time radar measurement techniques and adequately modeling the spatial extent of saturated areas at a relatively large scale. This work is a first step towards the latter goal.

Operational issues are also addressed. Models should be seen only as one component of a larger organization, within the framework of a decision-support system (Johnson, 1986). In a real-time flood forecasting system, modeling is a mean to aid decision making, not an end on itself.

Only a fully reliable forecasting model could work in isolation. Model and data imperfections introduce uncertainties in the forecasts which should be fully acknowledged and understood by decision makers. Traditional model development has been mainly concerned with model conceptualization: input data, physical variables, formulations, etc. This work is also concerned with model operation, based on the understanding that limitations in model behavior should be fully exposed to the user through a versatile and flexible interface, rather than hidden under a exclusive presentation of final results.

It is also important to locate the model within the broader realm of real-time decision making. The rainfall-runoff model should not be designed as a self-contained unit, a software package which only interacts with the end-user. There is an ever-increasing presence of computers in areas traditionally associated with human decision making. Modern operational flood warning schemes are complex systems, where the coordination of sensors, communications, data processing and modeling cannot always rely on direct human intervention. Complex systems are controlled by computers, and therefore hydrologic models intended to work in complex contexts should be designed in a way that facilitates their use by higher-level computer applications. They should be able to interface not only with human users, but also with other programs capable of making decisions. The problems of communication between different software modules in a real-time flood forecasting system are addressed here.

Software design for hydrologic models is an important concern of this work. The combination of recent advances in software development methodologies and interactive graphic capabilities of engineering

workstations constitute an adequate starting point to develop new ways to address the problem of real-time flood forecasting. The introduction of digital computers represented a revolutionary change in our ideas about hydrologic analysis (Snyder and Stall, 1965). Ever since, the evolution of computer modeling of catchment response has relied more on quantitative improvements in computer performance than on qualitative developments in model design. The way hydrologic models are designed has changed very little since the pioneering work on the Stanford Watershed Model. Functional analysis and structured programming are still the basic design methodologies applied to hydrologic model development, while other areas of computer simulation of physical systems have already evolved to object-based design methodologies (Widman et al., 1989). The application of these advanced design methodologies to hydrology can lead to more efficient and better structured modeling environments where the problem of real-time decision making can be addressed as a whole.

This work presents a software package for real-time flood forecasting, called Real-time Interactive Basin Simulator (RIBS). The package is based on a distributed rainfall-runoff model (Distributed Basin Simulator) intended to operate on midsize and large basins. Simplicity and computational efficiency for real-time performance have been the basic goals of model development. The model communicates with a hydrologic distributed database and presents its state, results and behavior to the user through an interactive graphic interface. The whole package is divided into different modules which communicate with each other under the control of a manager module.

1.2 Previous work on distributed modeling

The area of distributed modeling has been in constant evolution during the last 30 years. From the standpoint of flood forecasting, research has focused mainly on the basic problem of local runoff generation, which was originally approached as an infiltration phenomenon. The analysis of one-dimensional infiltration is presented first, and then the progressive evolution to more complex two and three-dimensional schemes is discussed.

Considerable literature exists on infiltration. Philip (1969) presents a detailed analysis of the mathematical treatment of the theory of flow through unsaturated porous media applied to the problem of infiltration. Skaags and Khaleel (1982) and Singh (1989) provide ample reviews of conceptual and empirical infiltration models, traditionally used to estimate rainfall excess under a one-dimensional or lumped approach.

The natural extension of the one-dimensional infiltration problem is the two-dimensional analysis of flow in a hillslope, where vertical infiltration combines with lateral flow to generate subsurface runoff. Interest in modelling subsurface flow in a hillslope in order to obtain stormflow arose from the realization that Hortonian runoff can only account for a fraction of the basin response observed in humid regions of high infiltration rate. Experimental studies of hillslope moisture profiles (Abdul and Gillham, 1989) or chemical hydrograph separation (Nolan and Hill, 1990; McDonnell, 1990) show that runoff-generation processes at the hillslope scale are more complicated than the basic infiltration-excess mechanism, and usually involve complex subsurface interaction between

'old' and 'new' water. That interaction, however, has proven difficult to characterize and "a detailed [...] understanding of hydrologic response [...] still eludes the hydrologic community" (Goodrich and Woolhisher, 1991).

When the spatial distribution of runoff generation is of concern, multidimensional models are required to evaluate the relative importance of vertical and lateral moisture flows. Zaslavsky and Rogowski (1969) called attention on the role of soil slope and anisotropy on the generation of lateral downhill flows and their impact on soil formation. Childs and Bybordi (1969) also analyzed the effect of soil layering on vertical infiltration. Zaslavsky and Sinai (1981a,b,c) presented a series of papers on the mechanisms that can produce lateral flow more or less parallel to the soil surface, illustrated with analytical models and numerical solutions. Two mechanisms are thought to have the largest effects: the surface transition layer and the existence of a layer of lower hydraulic conductivity.

The surface transition layer refers to the top portion of the soil where porosity varies from 100% in the air to a constant value at some depth. Higher hydraulic conductivities observed in this layer are usually explained in terms of macropores which run approximately parallel to the soil surface, providing a low-resistance path for lateral flow. There is clear evidence for the existence of flow through macropores (Hornberger et al, 1991), but modeling the phenomenon is extremely difficult, due to the complexities of the problem (Germann, 1990). The existence of a layer of lower hydraulic conductivity is also a factor that can produce lateral flow, since the layered soil as a whole acts as an anisotropic formation which deviates the flow from the vertical (Zaslavsky and Sinai, 1981b). McCord

and Stephens (1987) conducted tracer experiments to analyze water movement in the usaturated zone. They gathered evidence suggesting a strong lateral component of subsurface flow even in the absence of an impervious layer.

The numerous factors that affect unsaturated flow make it extremely difficult to model. For some simplified cases, analytical solutions can be obtained. Protopapas and Bras (1991a, b) obtained analytical solutions to the problem of two-dimensional unsteady infiltration, and found that, in the absence of slope, the importance of lateral flow in horizontally layered terrain is limited. Beven (1981, 1982a,b) presented an analytical model of subsurface stormflow generation based on the kinematic wave theory. Philip (1991a) provided an analytical solution to the problem of infiltration in sloped terrain in terms of infinite series, and extended his analysis to study concave and convex hillslopes (Philip, 1991b). He found that lateral flow is mostly driven by the combined effect of soil slope and anisotropy.

Although analytical solutions provide an excellent basis for understanding the influence of the diverse factors involved, their applicability to rainfall-runoff models is certainly restricted by the limitations of their assumptions. Factors such as soil heterogeneity, forcing functions or problem geometry can only be taken into account through analytical formulations, and therefore only simple cases can be treated.

In general, only numerical solutions to the three-dimensional flow problem are feasible, provided that there is enough knowledge about the boundary conditions and the spatially varying soil properties. Freeze (1971) presented a three-dimensional model based on the numerical integration of the partial differential equations governing overland and

subsurface flow. The model was calibrated to fit observed outflows and water table responses on a hillslope in the Reynolds Creek catchment in Idaho (Stephenson and Freeze, 1974). The calibration was evaluated by the authors as "less than perfect", although a "fairly complete" set of experimental measurements was available, and the scale of the application was very small.

In general, intensive data requirements prevent the application of numerical schemes to solve the three-dimensional governing equations, and simplifications must be adopted in order to keep the models operationally feasible at large scales. Simplifications are obviously related with the goals of the modeling effort, and represent relaxations of the basic assumptions made to model individual processes. They usually involve reducing the dimensionality of the problem, making assumptions about flow directions and discretizing the catchment into interacting one or two-dimensional components.

The Système Hydrologique Européen (Abbot et al., 1986a, b) is probably the more general application of distributed modeling to the land phase of the hydrologic cycle. The SHE is a physically-based model composed of a series of layered modeling units, all of them applied on the same basic orthogonal grid network. Modeling units include processes such as canopy interception, snowmelt, evapotranspiration, overland and channel flow, and unsaturated and saturated subsurface flow. Modeling layers are basically independent and can be run at different time steps. In addition to the independence of hydrologic processes, the strongest assumption of SHE refers to subsurface flow directions. It is assumed that flow in the unsaturated zone is essentially vertical, whereas flow in the saturated zone is essentially horizontal. Although the authors claim

general applicability, model structure and basic assumptions suggest applications at long time scales, mainly for continuous simulation of basin response.

Bathurst (1986a) reports on the application of SHE to the Wye catchment, a watershed of 10.55 km^2 in mid-Wales. Basic parameter values were derived from field measurements or published data and only those parameters to which the model is most sensitive were fixed by calibration. Calibration results were encouraging, although the inability of the model to account for lateral subsurface flow prevented an adequate fitting of both peak flow and baseflow recession. That aspect, together with the lack of physical basis to define initial phreatic surface levels, constitute the strongest limitations of the model.

The Institute of Hydrology Distributed Model (IHDM) is conceptually similar to the SHE (Beven, 1985), but places more emphasis on the definition of basin structure. Rather than following a rectangular grid, the catchment is divided into hillslope and channel components of irregular shapes, but the same basic processes are included in the simulation. Rogers et al. (1985) tested the model ability to reproduce observed streamflows in the Tanllwyth catchment (0.9 km^2) and conducted sensitivity analyses to assess the effects of uncertainties in parameter estimation. Surface roughness and hydraulic conductivity were identified as the most sensitive parameters, and model calibration was limited to them. Model performance was similar to that of SHE.

A significant body of research has been dedicated to investigate the role of topography in runoff generation. Dunne and Black (1970a, b) report on experimental studies on a hillside and a short reach of stream in the Sleepers River Experimental Watershed in Vermont. They found that the

major part of storm runoff is overland flow on saturated areas close to the stream which vary dynamically, both during the storm and seasonally. Subsurface flow is the key factor, because it maintains soil saturation during dry periods and accounts for the expansion of saturated areas during the storm, but its effect is largely controlled by topography at the basin scale. Saturated areas are likely to be found in convergent slopes, at the base of long hillslopes, and in areas of reduced soil moisture storage (Anderson and Burt, 1990a).

The prediction of the extension of the surface saturated areas has been the focus of numerous modeling efforts. Troendle (1985) describes models based on the variable source area concept. In the Variable Source Area Simulator, the watershed is divided into segments representing elementary hillslopes. Every segment is divided horizontally into increments and vertically into layers, and Richards equation is applied in a numerical scheme to route subsurface water between elements, thus accounting explicitly for subsurface lateral flow. Surface saturated elements configure the variable source area, where exfiltration takes place. The model was tested for several storms in two small forested watersheds (Fernow, of 38 ha, and Whitehall, of 24 ha). Results were better in the Fernow watershed, best suited for model application because of the uniformity and predictability of soil properties and configurations.

Data and computation requirements limit the applicability of detailed variable-source-area models. In order to minimize data requirements and to make models applicable to larger basins, several authors have adopted simplifying assumptions, generally based on topographic analysis, to incorporate the concept of variable contributing areas explicitly. O'Loughlin (1981) and Beven and Kirkby (1978) present models based on a

topographic wetness index that is dynamically used to define the surface-saturated area. These models are based on distributed information and are mostly physically based, but the introduction of the easily obtainable topographic index makes them very parsimonious and attractive. Moore et al. (1991) report studies by several other authors who found strong correlations between different topographic indices and soil moisture content, although they recommend caution when applying static indices to dynamic processes because of the hysteretic nature of the phenomenon. Hornberger at al. (1985) report on the calibration of a revised version of TOPMODEL (Beven and Wood, 1983; Kirkby, 1986) for a 5.15 km^2 catchment. It was found that, although the number of model parameters (13) was relatively low compared to other distributed models, hidden interdependence between sensitive and insensitive parameters prevented an adequate calibration.

Topographically-based models require a detailed representation of catchment relief. The increasing availability of digital terrain data has facilitated the expansion of this type of models, since automatic algorithms for terrain analysis reduce considerably the inconveniences of tedious manual methods. Moore et al. (1991) review the basic terrain modeling techniques. Three types of terrain data structures are available for computer processing: point elevation data on an irregular grid (Triangular Irregular Network), elevation data on a square grid and digitized contour data. Square-grid network data are of widespread use because of their ease of computer implementation and computational efficiency. Most recently developed distributed models adopt this representation, although the other two structures have also been applied successfully.

Moore et al. (1990) present the TAPES-C (Topographic Analysis Programs for Environmental Sciences - Contour) model, which is based on digitized contour lines. The model automatically partitions the basin into stream tubes, following equipotentials and streamlines of water flow according to topography. Several topographic attributes are computed automatically by the model, which are used to obtain lateral subsurface and return flow applying a surface-subsurface kinematic modeling approach. Output from the model includes runoff hydrographs and flow depths and velocities at any point in the basin and mapping of the zones of surface saturation. Application of the model to a 79.6 ha forested catchment produced good agreement with observed data for the calibration storm.

The widespread availability of digital elevation models on regular grids has focused the attention of investigators on the automatic processing of topographical information. The analysis of digital elevation models is currently an active area of research, with contributions ranging from the inference of basic topographic variables, flow paths and channel networks (Band, 1986; Morris and Heerdegen, 1988; Tarboton et al., 1991; Quinn et al., 1991) to the investigation of universal behavior in catchment geomorphology (Tarboton et al., 1989). Automatic algorithms to elaborate the topographic information contained in digital elevation models are convenient tools to describe and manipulate the terrain in distributed models.

Current research in distributed modeling can be grouped into two main categories. The first category consists of new model developments or modifications of pre-existing models. Adaptations of previously developed models usually address additional issues, such as water quality or

sediment transport (Woolhiser et al., 1990). Recent models published in the literature usually place emphasis on specific submodel components of special relevance for their modeling objectives. Blain and Milly (1991) presented a vertically integrated two-dimensional model of soil moisture which stressed the importance of lateral flow. The model by James and Kim (1990) concentrated on overland and channel flow, considering only infiltration-excess runoff. The possibility to obtain information from remotely sensed data has also affected model conceptualization. Ott et al. (1991) developed a distributed model where model parameters are directly derived from satellite imagery at a resolution of 30x30 m^2. The model gave excellent results when applied to a 18.5 km^2 test basin, apparently without calibration. Becchi et al. (1992) concentrated in the use of radar rainfall maps to predict the evolution of soil moisture content during flood events. They applied a distributed mass balance to account for moisture evolution and a scheme based on a network of linear reservoirs for channel routing. They tested the model at different spatial resolutions using synthetic storms.

The second category consists of contributions in which distributed models are used to explore the implications of changes in land management on basin response or to assess the effects of spatial variability on the performance of less complex, generally lumped, models (Gan and Burges, 1990a, b). Specially interesting in this group are the attempts to characterize subgrid variability in distributed models. Binley and Beven (1991) studied the prediction uncertainty of the IHDM using a very detailed three-dimensional Darcian flow simulator. Mancini et al. (1992) presented a similar analysis between a conceptual model based on a topographic index combined with Philip's infiltration equation and a

three-dimensional model of Richards equation in variably saturated porous media. Loague (1988) analyzed the effect of the spatial description of rainfall and soil hydraulic properties on the characterization of hillslope runoff. The stochastic-conceptual model designed by Freeze (1980) was the basis for this work. Binley et al. (1989a, b) used a fully three-dimensional subsurface flow model to assess the effect of different random patterns of saturated hydraulic conductivity on a small hillslope, of size comparable to that of a cell in a digital terrain model. They found that effective uniform hydraulic conductivities could be defined only in the case of high-permeability soils, although no consistent relationships could be found between the effective parameters and the moments of their distributions. Wood et al. (1988) applied TOPMODEL to address the issue of the relation between spatial heterogeneity and catchment scale. They analyzed the change of the statistical behavior of runoff generation with increases in catchment scale. They concluded that a Representative Elementary Area (REA) exists, as the minimum averaging unit from the standpoint of runoff generation.

Distributed modeling is still an unsolved area of research. Goodrich and Woolhiser (1991) summarize their review stating that "model evaluations which utilized observed data do not paint an encouraging picture of our ability to model catchment response". Significant improvements have been made possible by the development of computer technology, but it still remains unclear whether future increases in computer performance will solve all problems posed by distributed modeling. As Beven (1989) notes, "these problems result from limitations of the model equations relative to a heterogeneous reality; the lack of a theory of subgrid scale integration; practical constraints on solution

methodologies; and of dimensionality in parameter calibration". None of these problems can be addressed by a simple increase in computational power, however large. Practical issues, such as our ability to collect distributed data at the scale required by the models or to provide the necessary education for potential model users on a day-to-day basis, could also be added to the list if professional, widespread use of distributed models is to be achieved. Further research should address these problems.

1.3 Outline of approach

This work concentrates in two parallel lines of action: hydrologic model development and software engineering. It is believed that both approaches have benefited mutually from the interaction. Software design concepts have helped build a hydrologic model whose behavior can be easily analyzed and understood by the user in real time. On the other hand, the use of hydrologic concepts as basic entities in software analysis has led to a modular and efficient software package, which can be extended in many ways. This section presents the basic approach adopted to achieve the design objectives in both lines of action.

On the area of model development, the objective was to define a simple and computationally efficient rainfall-runoff model that could be used for flood forecasting in midsize and large basins. General research guidelines can be summarized in two aspects: (1) take advantage of radar-generated rainfall maps and (2) include topographical information in the form of digital elevation models. Chapter 2 is dedicated to the presentation

of the distributed rainfall-runoff model. A model structure based on the square grid of DEM's was selected. The basin is discretized in rectangular elements of homogeneous properties. Morphologic properties are extracted from the DEM and pedologic properties are obtained from a soil study. Model conceptualization is based on the analysis of distributed runoff-generation at the subgrid scale and the subsequent surface flow routing. Since uncertainty in the evaluation of rainfall excess usually dominates uncertainty in runoff prediction (Goodrich and Woolhiser, 1991), a greater emphasis was placed on the representation of runoff-generation processes rather than on flow routing.

Regarding runoff generation, two processes, infiltration-excess runoff and subsurface return flow, are included, since both contribute significantly to basin response in flood situations. The intended scale of application of the model required a simplified and computationally inexpensive model of infiltration which could use the topographical information provided by the DEM. Among the simplified models which have been developed for application in rainfall-runoff models, those based on the kinematic approximation (Beven, 1984; Charbenneau, 1984) offer a good platform to study the influence of slope, layering and anisotropy in gravity-dominated flow. The kinematic approximation neglects capillary potentials in the usaturated zone. For event-based models, concerned with fast basin response to intense rainfall, the penetration of the moisture wave is mostly controlled by gravitational forces and the effect of the capillary potential can be neglected, specially in the case of macropore-dominated flow (Anderson and Kneale, 1982). The kinematic infiltration model adopted is described in Chapter 2 and Appendix 1. Although one-dimensional in formulation, the model actually provides a description of

the two-dimensional features of subsurface flow in a hillslope. It accounts for the effects of terrain slope, anisotropy and soil layering under the influence of the gravitational potential. A simplified scheme to account for lateral moisture transfers between basin elements allows the representation of return flow.

Surface flow analysis is also based on computational simplicity. Schemes based on two-dimensional modeling of overland flow are computationally expensive and their applicability at large scales is questionable. Therefore, a simpler approach, based on travel times, was adopted. A routing scheme based on flow velocities has clear computational advantages, and its application is justified by the fact that the assumption of linear response is better for midsize and large catchments, specially at high flows. Both hillslope and stream travel velocities are considered by the model. The DEM is used to identify the stream drainage network based on the threshold area concept proposed by Tarboton et al. (1991). There is also the possibility to account for non-linearities in basin response through simple schemes.

The performance of the distributed basin simulator is analyzed in Chapter 3. In order to evaluate the relative importance of model parameters, a thorough sensitivity analysis was carried out. Emphasis was placed on verifying if the actual behavior of the model corresponded to the expectations and on studying the influence of model parameters on the different modes of runoff generation. Results of the sensitivity analysis were applied to calibrate the model for a midsize basin. Five events were used in a first calibration attempt, reserving other five events for a later evaluation step.

On the area of software engineering, the objective was to map the hydrologic analysis of the flood forecasting problem into a correlative software organization. General research guidelines have been taken from the permanently growing field of artificial intelligence, where software design methodologies have been developed within the context of general problem solving. The analysis of engineering problems in artificial intelligence usually leads to multilevel architectures, where symbolic and numeric methods are combined to achieve the final goal (Roddis and Connor, 1988). The system presented in this thesis, called Real-time Interactive Basin Simulator, is also structured in several levels.

The RIBS system architecture, presented in Chapter 4, is organized around the notion of generic task, as presented by Brown and Chandrasekaran (1989). A task is a combination of a problem, representation and inference strategy. Two types of agents are considered in a generic design problem: specialists and simulation tasks. Specialists address the problem of 'what to do', and are usually symbolic in nature. Simulation tasks address the problem of 'how to do it', and, in engineering contexts, are usually numeric in nature. Although system design takes into account the entire problem of real-time flood forecasting, the implementation work has focused almost exclusively in the simulation tasks, which incorporate the concepts of the distributed basin simulator. To allow for a smooth interaction of simulation tasks with different goals, an external manager module controls system inference in the absence of specialists.

Simulation tasks are implemented as separate modules which maintain open channels of communication with the control manager. An important feature of the overall design is the role of the central database,

which stores basin properties, states and model results. The general structure of modules operating on a central database follows the pattern of blackboard architectures (Nii, 1986). Although many distributed hydrologic applications rely on Geographic Information Systems for external data storage (Johnson, 1989; Sasowsky and Gardner, 1991), the goal of software portability, the relative homogeneity of the structure of the data involved and the need for fast transactions in real time are strong reasons that support the adoption of storage in direct-access files.

System implementation is discussed in Chapters 5 and 6. Chapter 5 describes the distributed basin simulator and Chapter 6 describes the user interface. Design and implementation have followed object-oriented techniques (Stefik and Bobrow, 1986; Cox, 1986; Booch, 1990). The main feature of RIBS implementation is the use of hydrologic concepts as software objects. In order to obtain maximum portability, the *C* language (Kernighan and Ritchie, 1988) was selected for implementation. Abstraction, encapsulation, inheritance and polimorphism are the main properties of object-oriented programming. Although *C* allows object-oriented programming, it does not support it explicitly. Therefore, RIBS implementation does not exhibit polimorphism, and inheritance was only considered in the design stage, since there are no mechanisms in *C* to implement it. These properties, however, were not essential in RIBS, because the class hierarchy is very simple.

The user interface is based on interactive graphic technology (Fedra and Loucks, 1985). Engineering workstations offer the possibility of a network-transparent window environment (Jones, 1989) which facilitates man-machine interaction. Each window is like a sheet of paper that can be moved, resized, stacked, put away and recovered. Windows can display

easy-to-understand graphic information, and the user can dynamically configure the screen to present the most relevant information at every moment. Object-oriented programming techniques also allow the definition of interactive interfaces (Young, 1990) where the user can establish a dialog with the model requesting the realization of specific tasks according to the situation. This possibility is used to provide multiple ways to access internal model representations, allowing the user to understand model behavior.

CHAPTER 2

The Rainfall-Runoff Transformation Model

This chapter presents a distributed, physically based, rainfall-runoff model that is later included in a real-time flood forecasting environment. The model is called Distributed Basin Simulator (DBS), and adopts the nodes in the grid of a Digital Elevation Model (DEM) as spatially discrete elements. Each node consists of a laterally homogeneous soil column in which vertical variability is parameterized following a simple scheme. The discrete spatial representation allows for the distributed definition of terrain slope, soil parameters and rainfall input, which are used to simulate the spatial distribution of runoff generation in the basin. The model is based mostly on topographical information, and the objective of the modeling process is to map the evolution of the saturated areas in the basin during a storm event. A simple routing procedure is applied to obtain basin response in points of the basin. After a description of the model representation of the basin, this chapter presents the runoff generation mechanisms and routing procedures adopted in DBS.

2.1 Basin representation

The three-dimensional geometrical structure of the basin is represented on a number of data layers which contain two-dimensional

information, as shown in Figure 2.1. Each layer is represented following the raster format proposed in the literature of spatial information systems. The layers are defined on a common rectangular grid composed of homogeneous elements. Raster format offers an excellent representation of spatially variable magnitudes if the number of grid elements is high. The rectangular grid representation was selected to

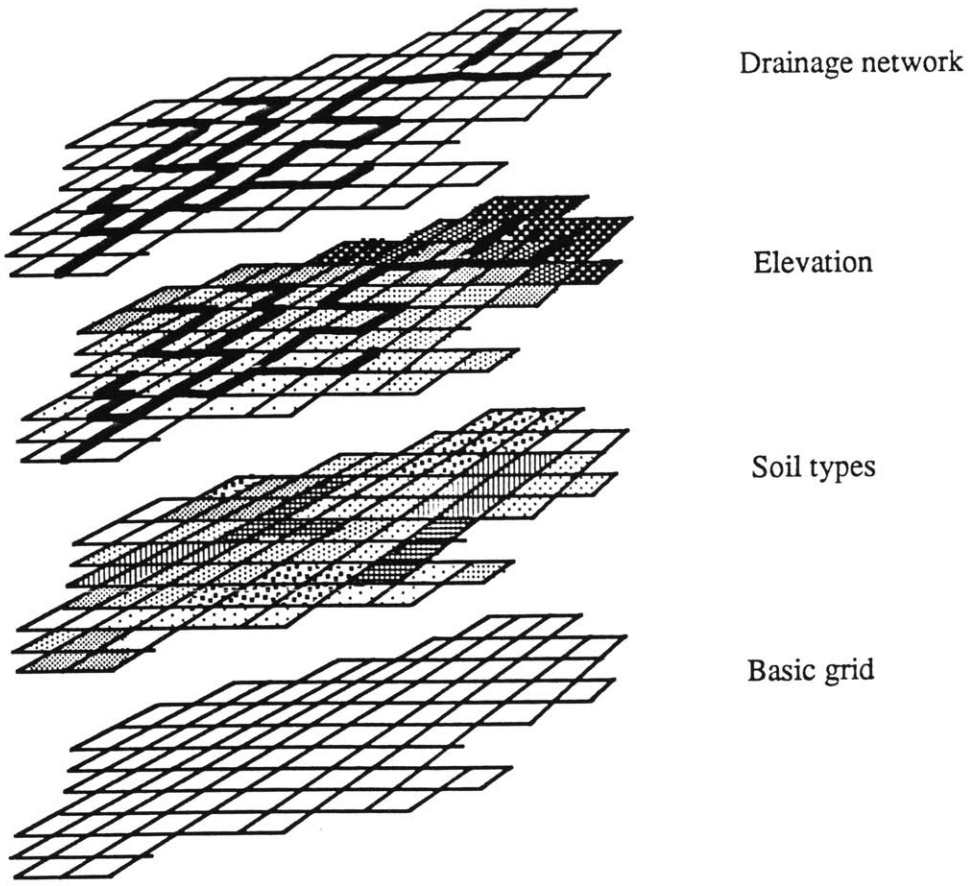


***Figure 2.1**: Representation of distributed information in the basin. All data layers share the same rectangular grid.*

match the detailed topographical information provided by DEM's. Other data used by the model may typically be available at lower spatial resolution, as in the case of rainfall intensity, or may be more naturally represented in vector format, as in the case of soil types or drainage networks, but the use of a standard raster format of equal size for all data layers offers clear advantages from the standpoint of computational efficiency in real time.

Two different scales of variability are combined in the model. Large scale spatial variability is represented through the distribution of different geomorphologic and pedologic properties on the nodes of the rectangular grid. The model also accounts for some variability at the subgrid scale, but only in the vertical direction. Model representation of the basin is therefore a number of points in the nodes of a rectangular grid (see Figure 2.1). Distributed variables are assigned values on the nodes of the grid, and are supposed to be representative of the square cell surrounding the node. Model equations are formulated in the vertical direction for the subgrid elements, which are assumed one-dimensional. Large scale interactions are taken into account through an element coupling scheme, which allows for moisture transfer between contiguous elements.

Every grid cell represents a portion of sloped soil. The representation of cell elements is shown in Figure 2.2 The geometry of the element is defined on three data layers: elevation, orientation and slope. The average elevation of the element as defined by the DEM is assigned to the element's central point. The slope orientation is approximated by the line pointing to the center of one of the eight cells surrounding the pixel. Connectivity between elements is based on slope orientation, which implements the relation "drains to". The slope value is given by the difference in

elevations divided by the horizontal distance. Techniques to analyze DEM's (Tarboton, 1989) are used to obtain orientations and slopes. The drainage network can also be obtained from the DEM assuming a threshold contributing area (Tarboton et al., 1989). Basin morphology is therefore completely defined in terms of its digital elevation map.

The second aspect of spatial representation refers to soil characteristics. Several soil properties are of interest to DBS. Each one is represented on an independent data layer, with values of the soil properties assigned to the nodes of the grid. Thematic cartography and field information can be used to generate spatial distributions of hydraulic conductivity, porosity and other soil properties of interest, but the investment required to obtain an accurate characterization of soil

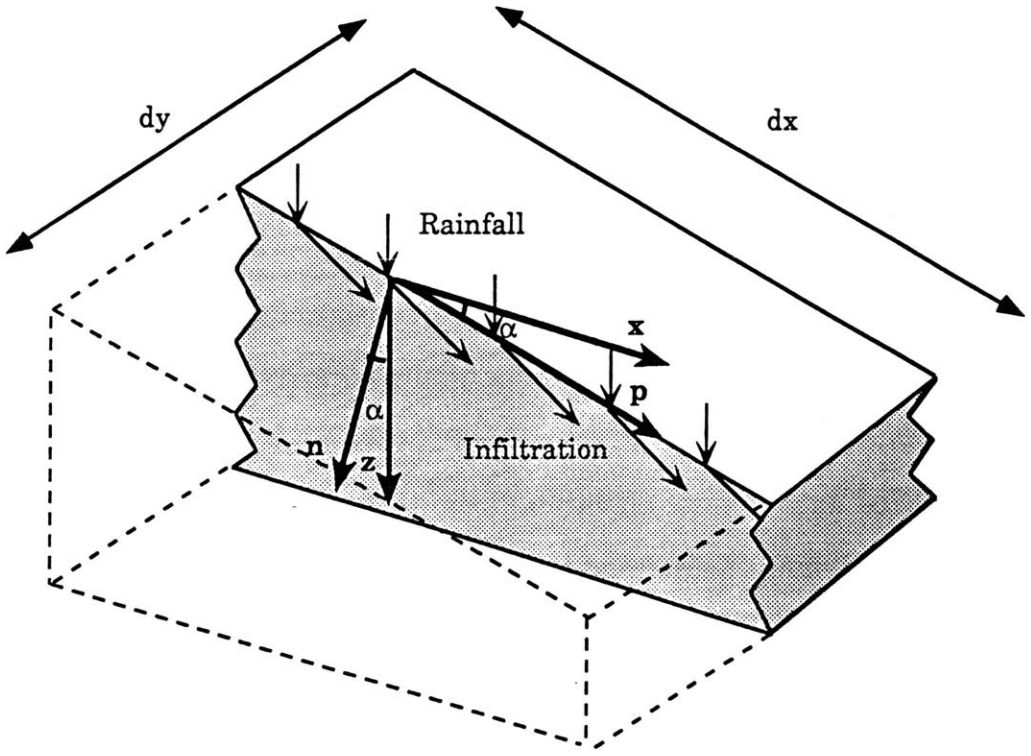


***Figure 2.2**: Model representation of a subgrid element.*

properties at the same resolution as the DEM is considerable. Therefore, model conceptualization emphasizes topographical information as the basis to analyze basin response.

The subgrid internal representation of the element is a vertically layered and laterally homogeneous sloped soil. The subgrid analysis is carried out in a reference system defined by the coordinate system *(n,p)* (Figure 2.2), where *n* follows the direction normal to the terrain slope (positive downwards) and *p* follows the direction parallel to the line of maximum slope (positive downslope). All variables within the element are assumed constant in the direction perpendicular to the plane *(n,p)*. Soil properties are considered to be homogeneous in the directions parallel to the terrain surface, but some variability is allowed in the direction perpendicular to the soil surface. The soil is layered, with normal hydraulic conductivity decreasing with normal depth. In order to keep model parameters to a minimum, a one-parameter analytical expression for vertical variability is assumed.

The model was designed to work on grids with thousands of elements, but the upper limit is only set by hardware limitations: memory allocation and execution time. For practical applications there is an optimum grid size, which is a function of the size of the catchment and of the variability of its properties. Currently available standard DEM's provide topographical data with horizontal resolution of 30 m. It is unlikely that, for practical applications, topographical data of higher resolution be widely available, and therefore 30 m is the typical scale of the lower spatial resolution considered in DBS. However, a spatial resolution of 30x30 m^2 would require a very large number of grid cells to represent

midsize catchments, of the order of hundreds of km^2, and therefore, larger grid sizes have to be considered for larger basins.

2.2 Runoff generation

The distributed basin simulator incorporates the two modes of storm runoff-generation mechanisms: infiltration excess and return flow. The basis for local runoff generation is the kinematic model of infiltration proposed by (Cabral et al., 1990). The model was further developed by the author, and the final version, adopted in this work, was finally presented by Cabral et al., (1992). For the reader's convenience, a copy of that paper is included in Appendix 1. This section presents an overview of the basic features of the kinematic model of infiltration and its application to the distributed basin simulator.

Although one dimensional in formulation, the kinematic model actually reproduces the two dimensional flow of water in an infinite slope of layered soil, and provides an expression to compute the net horizontal flow across a vertical cross section of the hillslope. At the basin scale, however, the subsurface flow of moisture is essentially three dimensional, due to the heterogeneities in slope orientation and soil characteristics. DBS approximates the real three dimensional flow by assuming the validity of the infiltration equations in each element at the subgrid scale and accounting for the transfer of moisture among cells in a simplified manner. The evaluation of runoff generation has two aspects: runoff generation at the subgrid scale using the one-dimensional model of infiltration and moisture transfer among elements.

2.2.1 The one-dimensional model of infiltration

The basin representation at the subgrid scale accounts for two main types of mechanisms that can generate runoff: infiltration excess, when rainfall intensity exceeds the infiltration capacity of the soil, and return flow, when there is convergence of subsurface flows in an already saturated area. In the first case, the runoff generation may be truly Hortonian, when the infiltration capacity of the soil column is equal to the saturated hydraulic conductivity at the surface, or it may be produced through a saturation-from-below type of mechanism, when a perched saturation zone reaches the surface reducing the effective infiltration capacity of the soil column. Return flow may be produced when subsurface flows converge into an area where perched saturation has developed.

The one-dimensional model of infiltration applied at the subgrid scale is described in this section. The basic assumptions regarding the representation of the soil column and the parameterization of vertical variability are reviewed first. Secondly, soil-water movement in the unsaturated and saturated areas is analyzed. Finally model equations for front evolution are presented.

Basic assumptions

The kinematic model considers a soil column of vertically heterogeneous, anisotropic and sloped soil in which the hydraulic conductivity decreases with depth. The terrain surface forms a slope

angle α with respect to the horizontal datum. Infiltration is described in the plane defined by the line of maximum slope and the normal to the terrain surface. Conditions are assumed to be uniform in the third spatial dimension, and variability in that direction is therefore neglected. The reference system is formed by the axes n and p, as represented in Figure 2.2. n is perpendicular to the terrain surface and positive downward, and p is parallel to the soil surface and positive in the downslope direction.

The flow of subsurface water is assumed Darcian. The full equations are considered in the saturated area, but the kinematic approximation (Beven, 1984) is adopted in the unsaturated zone, where the contribution of capillary pressure to the hydraulic gradient is neglected. Under the kinematic approximation, the infiltration capacity of a soil column is equal to the saturated hydraulic conductivity at the surface. A rainfall intensity R, at the surface, higher than the saturated hydraulic conductivity, K_{sat}, would produce a Hortonian infiltration-excess runoff equal to R - K_{sat}. Rainfall of intensity lower than K_{sat} will infiltrate completely during the early stages of the storm, leading to a moisture distribution along the soil profile capable of maintaining an infiltration rate equal to R. However, heterogeneities in the vertical distribution of hydraulic conductivity may lead to the formation of a zone of perched saturation, whose top boundary ascends as the storm progresses, and may eventually reach the surface producing a "saturation-from-below" type of runoff generation.

The Brooks-Corey parameterization scheme (Brooks and Corey, 1964) is adopted to relate the unsaturated hydraulic conductivity and the pore pressure with moisture content. The soil is anisotropic, with main directions of anisotropy parallel to the axes n and p. The saturated

hydraulic conductivities in the main directions are considered to decrease with normal depth. In order to obtain analytical formulations of the model, the variation of saturated hydraulic conductivity with depth is limited to the exponential case:

$$K_{S_n}(n) = K_{0_n} e^{-fn} \tag{2.1a}$$

$$K_{S_p}(n) = K_{0_p} e^{-fn} \tag{2.1b}$$

where $K_{Sp}(n)$ and $K_{Sn}(n)$ are the saturated conductivities at depth n perpendicular to the surface; K_{0n} and K_{0p} are the saturated hydraulic conductivities in directions n and p at the soil surface; and f is a parameter of dimension $[L^{-1}]$, which controls the decay of the saturated hydraulic conductivity with depth.

Other soil properties (anisotropy, porosity, etc.) are considered homogeneous within each element, although they are allowed to vary throughout the basin. The saturated hydraulic conductivities in directions p and n are related through the dimensionless anisotropy ratio a_r, defined as

$$a_r = \frac{K_{0_p}}{K_{0_n}} > 1 \tag{2.2}$$

This relationship is assumed to be valid for all depths. For unsaturated soils, hydraulic conductivity is a function of moisture content. Upon substitution of Equations (1a) and (1b) for the saturated conductivities in directions n and p, the Brooks-Corey (Brooks and Corey, 1964) parameterization gives

$$K_n(\theta,n) = K_{0_n} e^{-fn} \left(\frac{\theta - \theta_r}{\theta_s - \theta_r} \right)^{\varepsilon} \qquad (2.3a)$$

$$K_p(\theta,n) = K_{0_p} e^{-fn} \left(\frac{\theta - \theta_r}{\theta_s - \theta_r} \right)^{\varepsilon} \qquad (2.3b)$$

where $K_n(\theta,n)$ and $K_p(\theta,n)$ are the hydraulic conductivities in directions n and p at moisture content θ and at depth n; θ_s is the saturated moisture content; θ_r is the residual moisture content, defined as the value below which moisture cannot be extracted by capillary forces; and ε is a pore size distribution index.

Unsaturated flow description

Model dynamics are based on the kinematic approximation (Beven, 1984; Charbeneau, 1984) for flow in the unsaturated area. Under the kinematic approximation, the capillary pressure gradient is neglected and as a result moisture waves during a storm event can be described as sharp discontinuities that separate areas of different moisture content. These discontinuities are usually referred to as fronts, and are assumed to proceed perpendicular to the terrain surface in homogeneous or layered soil. Two fronts are considered in the kinematic model described here: a wetting front which represents the penetration of the moisture wave into the soil and a top front which represents the ascent of the perched saturation zone that develops when moisture flux in the unsaturated area is greater than hydraulic conductivity at the wetting front. The normal

depths of the wetting and the top fronts are N_f and N_t respectively, as represented in Figure 2.3.

The total moisture content of the cell above the wetting front M_t is divided into an unsaturated and a saturated area. Unsaturated moisture content is M_u and saturated moisture content is M_s. The relations between moisture contents and front positions are given by the equations:

$$M_t = \int_0^{N_f} \theta(n)\, dn \qquad (2.4a)$$

$$M_u = \int_0^{N_t} \theta(n)\, dn \qquad (2.4b)$$

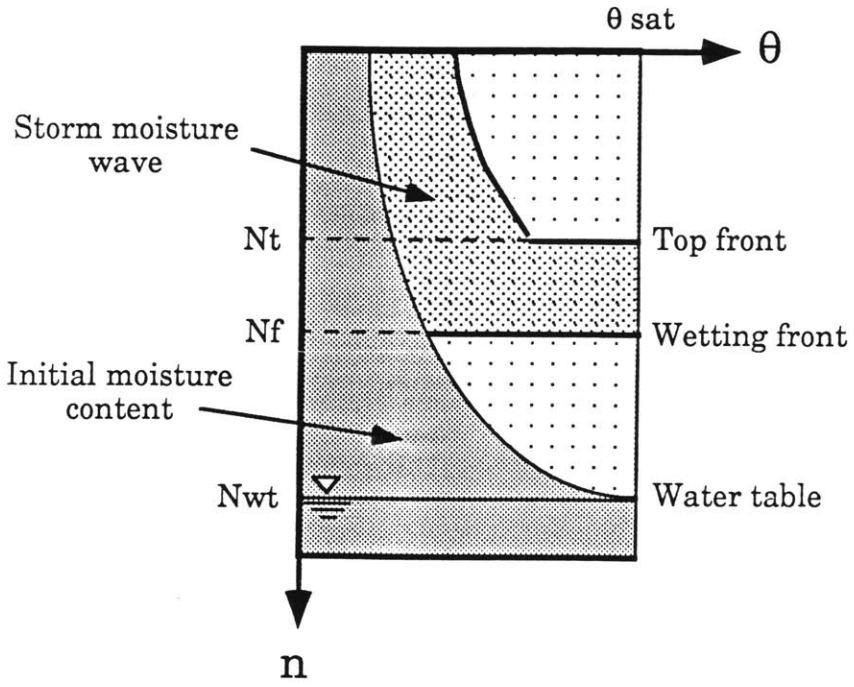


***Figure 2.3**: Schematic representation of the wetting and the top fronts. The horizontal axis represents moisture content θ, and the vertical axis represents normal depth.*

$$M_s = \int_{N_t}^{N_f} \theta(n)\, dn \qquad (2.4c)$$

$$M_t = M_u + M_s \qquad (2.5)$$

The model accounts for the evolution of moisture in the saturated and the unsaturated zones. The unsaturated area extends from the terrain surface to the top front, and the saturated area extends between the wetting and the top fronts. Moisture distribution is defined according to the simplifying assumption of constant normal flux in the unsaturated area, implying that flux variations in the normal direction are quickly smoothed out. This assumption is consistent with the one-dimensional continuity equation for the unsaturated zone:

$$\frac{\partial \theta}{\partial t} + \frac{\partial q_n}{\partial n} = 0 \qquad (2.6)$$

Negative flux gradients in the normal direction imply local moisture accumulation, which in turn increases the hydraulic conductivity of the soil and correspondingly increases the local normal flux, reducing the gradient.

The assumption of constant normal flux in the unsaturated area leads to the description of the soil moisture profile under steady infiltration at rate R. As described in Appendix 1, the unit gravitational gradient leads to the expression $R = K(\theta,n)$. Substituting for $K(\theta,n)$ according to the Brooks-Corey parameterization and solving for θ

$$\theta(R,n) = \left(\frac{R}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r)\, e^{\frac{f}{\varepsilon}n} + \theta_r \qquad (2.7)$$

Equation (2.7) shows that for constant normal flux, the moisture content above the wetting front increases exponentially with depth, in order to keep the normal hydraulic conductivity equal to the infiltration rate R.

Given that saturated soil conductivity decreases with depth, for a certain depth within the soil profile, $N^*(R)$, the saturated conductivity in the normal direction will be equal to the rainfall rate R (for the case $R < K_{0n}$), that is

$$K_{S_n}(N^*) = R \qquad (2.8)$$

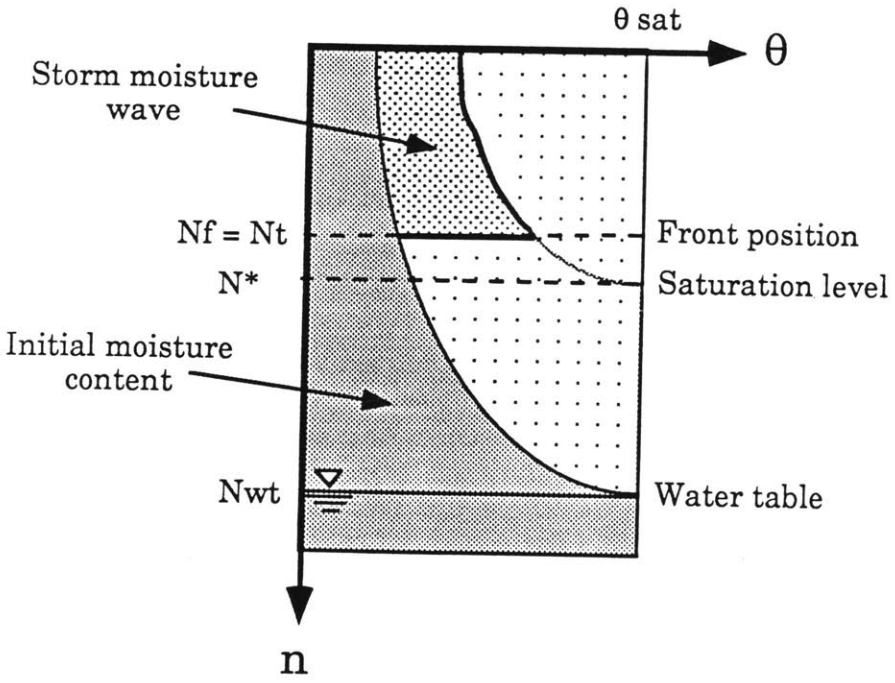


***Figure 2.4**: Schematic representation of unsaturated flow. Both fronts coincide. The moisture content above the fronts is given by equation 2.7.*

Substituting the expression for $K_{s_n}(N^*)$ obtained from Equation (2.1b) into equation (2.8) and solving for N^* we obtain,

$$N^*(R) = \frac{1}{f} \cdot \ln\left(\frac{K_{0_n}}{R}\right) \qquad (2.9)$$

Equation (2.9) applies only for $R \leq K_{0n}$. For $R > K_{0n}$, the saturated level is at the surface, and there is no unsaturated area above the wetting front.

Making $n = N^*(R)$ in Equation (2.7), we obtain $\theta(R,N^*) = \theta_s$. Therefore, $N^*(R)$ represents the depth at which saturation develops under a steady infiltration rate R. For $n > N^*(R)$, we have $K_{s_n}(n) < R$ and the soil can no longer transmit flow at the rate of infiltration to depths beyond $N^*(R)$. Water accumulates above that level and perched saturation develops. Figure 2.4 represents schematically the concepts related to unsaturated flow.

Saturated flow description

The soil column above the wetting front is saturated in two cases; 1) when the rainfall rate is higher than the surface saturated conductivity ($R > K_{0n}$), or 2) when the wetting front has penetrated beyond the critical depth ($N_f \geq N^*(R)$).

In the first case, the entire wetted soil (from the surface to the wetting front) is saturated. In the second case, as the wetting front reaches $N^*(R)$, normal flux below the saturation level is less than recharge from above, and moisture progressively accumulates above the wetting front. A

zone of perched saturation develops and grows upward from $N^*(R)$, as well as downward as the front progresses.

The first term in the continuity equation (2.6) is zero within the zone of saturation, since $\theta(t) = \theta_S$, and the saturated flow is therefore non-divergent, which in one dimension means:

$$\frac{\partial q_n}{\partial n} = 0 \qquad (2.10)$$

According to Equation (2.10) , $q_n(n)$ is constant in the n direction. Since the elevation gradient is constant and hydraulic conductivity decreases with depth, constant normal flow within the saturated zone implies a positive pressure buildup in it. Pressure gradient compensates for the different hydraulic conductivities of the successive layers of the saturated zone in order to keep normal flow constant. Therefore, the hydraulic gradient within the saturated zone has to account for the gradient of that positive pressure distribution.

The pressure distribution can be obtained assuming that at both fronts, N_t and N_f, pressure is atmospheric. The resulting pressure distribution is (see Appendix 1 for details)

$$\Psi(n) = \cos(\alpha)\left[\left(n + \frac{1}{f}\right) - \frac{e^{fN_f} - e^{fn}}{e^{fN_f} - e^{fN_t}}\left(N_t + \frac{1}{f}\right) - \frac{e^{fn} - e^{fN_t}}{e^{fN_f} - e^{fN_t}}\left(N_f + \frac{1}{f}\right)\right] \qquad (2.11)$$

The hydraulic potential can be obtained upon differentiation of the pressure distribution. After substitution of the normal component of the hydraulic potential in the flow equation, we get an expression for the normal flow in the saturated area,

$$q_n = K_{0_n} \frac{f(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \cos(\alpha) \qquad (2.12)$$

Normal flow is constant in the saturated area. We can define an "equivalent hydraulic conductivity" for the saturated area, K_{eq}, as the normal hydraulic conductivity of a homogeneous soil with the same normal flow q_n given by Equation 2.12. The equivalent hydraulic conductivity is given by

$$K_{eq}(N_f, N_t) = K_{0_n} \frac{f.(N_f - N_t)}{e^{f.N_f} - e^{f.N_t}} \qquad (2.13)$$

The equivalent conductivity corresponds to the harmonic mean of the conductivities over the saturated depth,

$$K_{eq}(N_f, N_t) = \frac{\int_{N_t}^{N_f} dn}{\int_{N_t}^{N_f} \frac{dn}{K_{ns}(n)}} \qquad (2.14)$$

We may also designate by "equivalent depth", N_{eq}, the normal depth which has saturated hydraulic conductivity equal to $K_{eq}(Nf,Nt)$. From Equations (2.1a) and (2.13),

$$N_{eq}(N_f, N_t) = -\frac{1}{f} \ln\left[\frac{f.(N_f - N_t)}{e^{fN_f} - e^{fN_t}}\right] \qquad (2.15)$$

N_{eq} is also the depth at which the pressure distribution (Equation (2.11)) is maximum, because for that point the pressure gradient is zero and flow is controlled only by the gravitational gradient. For depths smaller than N_{eq}, that is $N_t < n < N_{eq}$, the saturated hydraulic conductivity is greater than K_{eq}, and the pressure gradient is positive (increasing pressure with depth) to compensate for the excess in hydraulic conductivity and keep the normal flow constant. For depths greater than N_{eq}, that is $N_{eq} < n < N_f$, the saturated hydraulic conductivity is smaller than K_{eq}, and constant normal flow implies a negative pressure gradient in that area, up to the wetting front, where pressure is again atmospheric. Figure 2.5 represents moisture and pressure distributions in the saturated area.

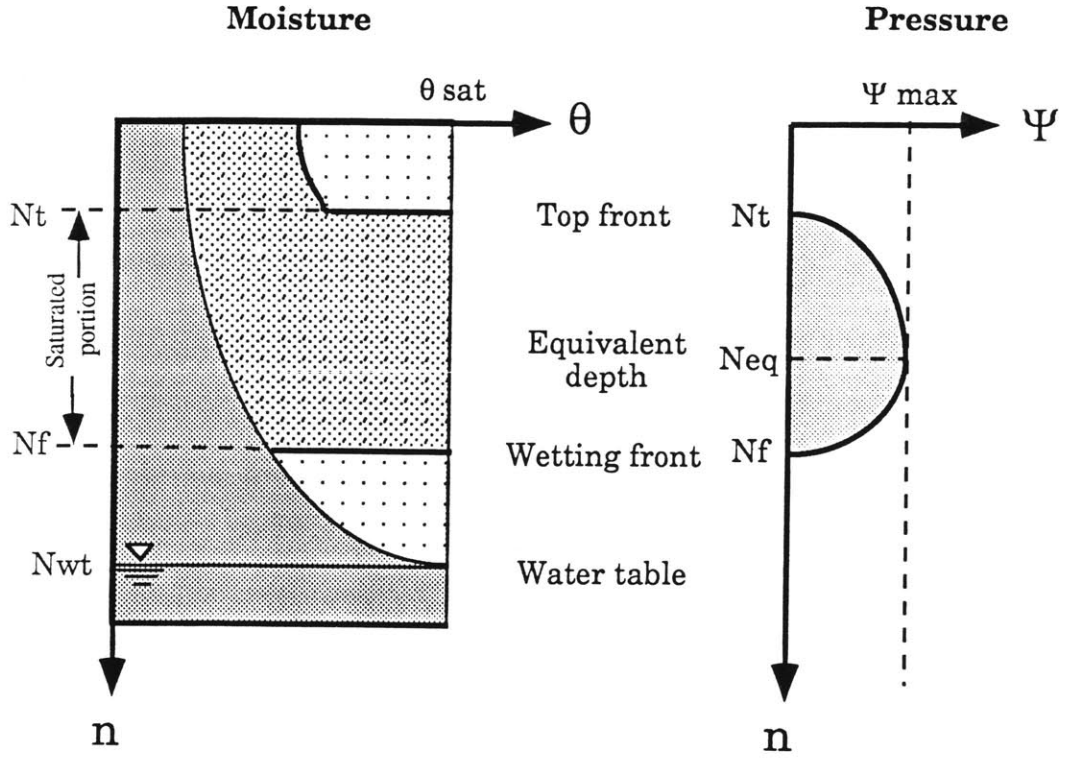


***Figure 2.5**: Schematic representation of saturated flow. Positive pressure develops between the wetting and the top fronts. The conductivity of the soil at the point of maximum pressure corresponds to the equivalent hydraulic conductivity of the saturated area as a whole.*

Front evolution equations

Three state variables (the moisture content, the wetting front depth and the top front depth) define the moisture state in the soil column. Below the wetting front the moisture profile is that corresponding to the initial state before the beginning of the storm. Between the wetting and the top fronts the soil column is saturated. If the element has not reached saturation yet, the wetting and the top fronts coincide and the saturated volume is null. Above the top front the moisture is distributed with the condition of uniform normal infiltration. The kinematic model provides equations that describe the dynamics of the state variables. Subsurface moisture inflows to the soil column are given as upstream (i.e. upslope) boundary conditions to the problem. Moisture outflows depend only on the internal distribution of moisture along the soil column, and therefore, once the values of the state variables are known, the horizontal moisture outflow from the cell can be computed. Surface runoff can then be evaluated as a function of rainfall intensity, saturated hydraulic conductivity at the surface, saturation level depth and moisture balance in the pixel.

The evolution of the state variables is described by a system of three first-order differential equations. Model equations are derived in Appendix 1. Front evolution is described by two basic set of equations, each one corresponding to a possible moisture profile, depending on whether a perched saturation area has developed between the top and the wetting front. The equation for the evolution of the third state variable, moisture content, is obtained through mass conservation considerations. A summary of the model formulation is presented hereof.

When the wetting front is not saturated both wetting and top fronts coincide, and their evolution is controlled by the following equation:

$$\frac{dN_f}{dt} = \frac{(R - R_i)\cos(\alpha)}{\theta(R, N_f) - \theta(R_i, N_f)} \qquad (2.16)$$

where

- N_f is the wetting front normal depth, which is also equal to the depth of the top front N_t for unsaturated infiltration
- α is the slope angle
- R is the equivalent rainfall intensity, which is equal to the normal infiltration flux in the unsaturated portion of the soil column. When rainfall intensity is variable, R represents an equivalent constant rainfall intensity that would lead to the same moisture distribution in the soil column
- R_i is the initial recharge rate, which describes the moisture state at the beginning of the storm
- $\theta(R,n)$ is the moisture content of the soil column at depth n when the infiltration rate is R

The model assumes an exponential decrease of hydraulic conductivity with depth, and therefore, as the front progresses downward it reaches levels of lower permeability. To maintain a normal infiltration flux equal to R, moisture content must increase with depth. Eventually, moisture content reaches saturation, and the governing equations change. A zone of perched saturation develops, which grows both upwards and downwards because normal flux in the unsaturated zone

(equal to the rainfall intensity R) exceeds infiltration capacity in the saturated area.

The equations for the evolution of the wetting and the top front of the perched saturation zone are:

$$\frac{dN_f}{dt} = \frac{q_n - R_i\cos(\alpha)}{\theta_s - \theta(R_i, N_f)} \qquad (2.17a)$$

$$\frac{dN_t}{dt} = \frac{q_n - R\cos(\alpha)}{\theta_s - \theta(R, N_t)} \qquad (2.17b)$$

- N_t is the top front normal depth
- θ_s is the saturation moisture content
- q_n is the normal infiltration flux in the saturated portion of the soil column, given by Equation (2.12)

When the storm moisture wave reaches the water table it can only be drained laterally, since no vertical moisture flow can occur below the water table. Water table levels in the basin are in equilibrium draining the initial recharge rate R_i, and therefore normal infiltration capacity at the water table is equal to the initial recharge rate. When the wetting front reaches the water table depth, it can no longer proceed downwards, and the rate at which moisture accumulates above it exhibits a sharp increase. The eventual effect is a local rise of the water table in that element which will make it completely impervious if the saturated area reaches the surface.

2.2.2 Adaptation to basin scale

The one-dimensional model of infiltration is formulated for conditions which differ significantly from those under which the basin scale model is intended to operate. The infiltration model is defined for a constant rainfall intensity in a uniform slope of infinite length. Water movement occurs in the plane defined by the vertical direction and the line of maximum slope, and, since the slope is infinite, lateral flow for any vertical section is balanced. DBS is a basin-scale model, and the infiltration model is applied to rectangular grid elements, limited in size. Every element receives subsurface flow from upstream elements and transmits its own contribution to the element downstream of it. There are also effects of flow aggregation. As a result, lateral inflow and outflow do not necessarily coincide for a given pixel. Front position will vary from pixel to pixel, since soil properties and pixel slope will change from point to point in the basin. Rainfall intensity will also vary in time and space. Therefore, several additional assumptions and modifications have to be made in order to adapt the infiltration model to variable rainfall, unbalanced lateral flow and spatial heterogeneities. These modifications play an important role on the practical implementation of the model, because they affect the numerical accuracy and stability significantly. This section concentrates on the problem of variable rainfall rates. The problems of unbalanced lateral flow and spatial heterogeneities affect the moisture balance equation, and are discussed in detail in Section 2.2.3.

In order to adapt the model to variable rainfall rates, the following assumption is made: water gets redistributed in the normal direction in order to attain uniform normal flow. This means that only a single

moisture wave will propagate downwards, regardless of the variability of rainfall intensity during the storm. This is a strong assumption, which is to some extent supported by the dynamics of the unsaturated infiltration mechanism, which, at least qualitatively, tend to redistribute moisture. For the uniform rainfall case, moisture content is such that the unsaturated hydraulic conductivity equals the normal infiltration flow. For the variable rainfall case, an average infiltration flow can be defined. If moisture content at some level is higher than that corresponding to the average flow, the hydraulic conductivity is higher than the average, and moisture will tend to migrate from that point. Conversely, if moisture content is lower, water will tend to accumulate in that area, increasing hydraulic conductivity until an equilibrium is reached.

The solution adopted is to define an equivalent uniform rainfall rate that would lead to the same moisture content in the unsaturated portion of the pixel. The moisture profile is given by Equation 2.7. Integrating the moisture profile above the wetting front for an equivalent rainfall rate, R_e and equating it to the unsaturated moisture content, M_u, we obtain

$$\int_0^{N_t}\left[\left(\frac{R_e}{K_{0n}}\right)^{\frac{1}{\varepsilon}}(\theta_s-\theta_r)\,e^{\frac{f}{\varepsilon}n}+\theta_r\right]dn=M_u \qquad (2.18)$$

Solving for R_e

$$R_e=K_{0n}\left(\frac{M_u-\theta_r N_t}{(\theta_s-\theta_r)\frac{\varepsilon}{f}(e^{\frac{fN_t}{\varepsilon}}-1)}\right)^{\varepsilon} \qquad (2.19)$$

The expressions for front evolution are assumed still valid, substituting R by the equivalent rainfall rate R_e. Equation (2.19) is only valid when there is an unsaturated area in the pixel, that is when $N_t > 0$. When the top front is at the surface, the equivalent rainfall rate is equal to the actual rainfall rate at that time.

2.2.3 Moisture balance

Once the position of the fronts is known, the moisture distribution in the soil column can be obtained. Since gradients are also known, moisture fluxes at every point can be computed applying Darcy's equation. Moisture outflows are in turn a function of moisture fluxes. The equations governing vertical and lateral moisture flows are obviously coupled. To account rigorously for moisture transfers among elements in the grid, the full three-dimensional equations of moisture flow should be applied, accounting for soil heterogeneities both in the vertical and in the horizontal directions. That approach introduces computational complications which make it impractical for application at large basin scales. Therefore, simplifications must be made in order to obtain a computationally efficient model that can operate in real time for basins composed of a large number of elements.

Simplifications in DBS are based on the idea of decoupling vertical and horizontal moisture equations. That approach is supported by the fact that the forcing mechanisms driving vertical and lateral movement of moisture during a storm are essentially different in nature and operate at different time scales. Vertical movement is largely based on the gravitational forcing for significant infiltration rates. The vertical

redistribution of moisture due to capillary forces is only of secondary importance and operates at a much slower time scale. For the lateral movement, however, there is no gravitational forcing, and capillary forces are predominant, transporting water from wet to dry areas in the basin. In any case, gravitational forces are predominant in the overall water transport. Moisture concentrates on troughs and depressions, in a general downslope movement following topography, and that is the most important feature that should be captured by the model.

Two types of moisture transfers among elements are considered in DBS. First, the one-dimensional model of infiltration applied at the subgrid scale predicts deviations of flow from the vertical even in the case of laterally homogeneous terrain. When the model is applied to a bounded domain, the horizontal component of flow produces a net flow of moisture at the downslope boundaries, which is transmitted to the contiguous element. In addition to that, the application of the gravity-dominated one-dimensional model of infiltration to every element independently gives different pressure and moisture distributions in every cell, which in turn lead to horizontal pressure and moisture gradients that drive lateral flow between pixels. In the former case, moisture transfers among pixels are a consequence of the state in every element, and in the latter, moisture transfers are a consequence of the difference in moisture profiles in contiguous elements. Both moisture transfers are taken into account in a simplified manner, described as follows.

Lateral flows in homogeneous terrain

In the kinematic model of infiltration, local terrain slope, heterogeneity and anisotropy produce a diversion from the vertical for the infiltration in the saturated zone. Consequently, a lateral movement of moisture may exist. The model was originally defined for an infinite homogeneous hillslope, and therefore it was not affected by boundary conditions. Model hypotheses are not valid when it is applied to a finite domain, like the cells considered in DBS, because boundary conditions will in general affect the patterns of circulation of subsurface flows. However, since no simple solutions are available for the full three-dimensional equations under arbitrary boundary conditions, a simplified element coupling scheme was adopted to account for moisture transfers between elements.

It is considered that the one-dimensional infiltration equations are valid at the subgrid scale. The equations describe the two dimensional flow contained in a plane defined by the normal to the terrain surface and the line of maximum slope. The horizontal component of the flow can be integrated along any vertical surface to obtain the net flow across that boundary. In particular, according to the description of flow given in Section 2.2.1, the total subsurface outflow from the wetted soil, represented by Q_{hout}, from a vertical cross-section of width W is obtained through integration over the wetted depth (i.e. from 0 to $\frac{N_f}{cos(\alpha)}$) of the horizontal component of the moisture flow in the soil column, as shown in Appendix 1. The resulting expression is

$$Q = W \sin(\alpha) \left\{ [N_t R (a_r - 1)] + \left[K_{0_n} \frac{a_r}{f} (e^{-fN_t} - e^{-fN_f}) \right] - \left[K_{0_n} \frac{f (N_f - N_t)^2}{e^{fN_f} - e^{fN_t}} \right] \right\} \quad (2.20)$$

For a rainfall rate lower than the initial infiltration capacity (for $R < K_{0n}$) lateral discharge is given only by the first term of (2.20) while $N_t = N_f < N^*(R)$. After perched saturation has developed (for $N_t < N_f$) lateral discharge is given by all three terms of equation (2.20) while $N_t > 0$. However, eventually $N_t = 0$, and lateral discharge is then given only by the second and third terms of Equation (2.20).

This expression for the horizontal subsurface flow across a vertical cross section is applied to evaluate outflows from all elements in the basin. Subsurface inflows into a certain element are given by the sum of the outflows from all the upstream elements draining directly into it. In order to account for moisture transfers between elements correctly, the computations should be carried out recursively, according to the relation "drains to". This guarantees that outflows from all upstream elements have already been computed before considering a given element, and therefore the total subsurface inflow to the element is available.

Two major simplifications are implicit in this scheme. First, the coupling effect between elements is only indirectly considered. Each pixel is represented by model equations applied to its central point. Model equations consider it to be an effective infinite extension of soil, with no boundary effects. The coupling between elements is taken into account in the mass conservation equation, but it is neglected in the momentum equation. Every pixel receives moisture from the upstream pixels draining into it. That moisture is taken into account in the mass balance, but it is considered that these flows do not affect the momentum equation,

that is, the speed at which fronts move. Since the momentum equation is formulated in the direction normal to the slope, the approximation may be acceptable as long as the volume of water transferred is only a small fraction of the total volume of water stored in the element.

The second simplification refers to the assumption about flow geometry. The spatial orientation of the flows entering every cell is assumed parallel to the line of maximum slope, irrespective of the orientation of the slopes of the upstream pixels. This assumption is strong and unreal considering the idealized model adopted for the description of basin terrain, where sharp discontinuities are defined at the boundaries between pixels, but it is not so strong considering the situation in the real basin, where transitions are usually smooth and subsurface flows tend to be parallel to the lines of maximum slope, following the general topography of the area.

Lateral flows resulting from the spatial variability

Since different pixels have different moisture and pressure distributions in the vertical direction, there are horizontal moisture and pressure gradients which drive lateral moisture flows among the pixels. Lateral gradients are relatively small, because for a typical pixel size horizontal distances between two consecutive elements are very large compared to vertical distances. Therefore, lateral flows due to spatial variability of moisture are small compared to lateral flows due to topography and anisotropy, as shown by analyses of moisture distributions in real basins (Dunne et al., 1975; Tanaka et al., 1988, Gbureck, 1990).

A very simplified procedure is applied to account for the effect of horizontal pressure gradients on lateral flow. It is considered that the equations describing front evolution and moisture distribution in every element are valid for the central point of the grid cell. The one-dimensional model provides therefore an estimate of water pressure distribution along the normal to the surface of every element in its central point. Since different pixels have different pressure distributions, the hypothesis of lateral uniformity made in the one-dimensional model of infiltration is not satisfied, because the terrain is not infinite in practice, and at some point, boundary effects affect the solution. To account for this in a simplified form, the method of superposition of solutions was adopted. Model equations are applied to every element independently, and the results are then corrected accounting for moisture transfers due to lateral imbalances of pressure between two consecutive pixels. The correction applied is detailed next.

The objective is to estimate the horizontal moisture flow between two contiguous pixels as a consequence of pressure imbalance. The use of the full equations of Darcian flow is complicated by the geometry of the problem. The saturated area may be located at different depths in contiguous pixels and the normals to the surface may not be parallel if terrain slopes are different. The model adopted to describe the terrain is also unrealistic, with spatially homogeneous soil of different type in every cell separated by a sharp discontinuity. Furthermore, additional complications are introduced by the different relative positions of contiguous pixels within the grid.

The geometry of the problem is considerably simplified if the difference of the inclinations of the normals with respect to the vertical is

neglected. That is equivalent to considering uniform slope in both pixels. That slope may be given by the difference in elevation of the central points of the two consecutive pixels divided by the horizontal distance, and is not a bad approximation in most cases. Under that approximation, the total lateral gradient between two contiguous pixels is given by

$$J_x(z) = \frac{\partial \Psi}{\partial x} \approx \frac{\Delta \psi(z)}{\Delta x} = \frac{\psi_2(z) - \psi_1(z)}{x_2 - x_1} \qquad (2.21)$$

where Ψ is the pore pressure, z represents vertical depth and x, horizontal distance. $\frac{\partial \Psi}{\partial x}$ may be approximated as $\frac{\Delta \Psi(z)}{\Delta x(z)}$. Neglecting the effect of slope in Δx (Front depths are of the order of a few meters at the most, while horizontal distances between grid points are of the order of tens of meters), Δx is independent of z. Also, since we have assumed that both elements have the same slope, the vertical depth z can be substituted by the normal depth $\frac{n}{cos(\alpha)}$, where α is the average slope angle of both elements.

Lateral flow is given by Darcy's equation

$$q_x(z) = -K_{eq}(z)\, J_x(z) \qquad (2.22)$$

The hydraulic gradient is given by Equation (2.21). For the equivalent hydraulic conductivity $K_{eq}(z)$, some average of the hydraulic conductivities of both elements has to be used. The selection is complicated by the anisotropy and heterogeneity, both in the vertical and in the horizontal, of the soil. Since they are connected in series, the geometric

mean of the saturated parallel hydraulic conductivities is considered an acceptable approximation

$$K_{eq}(z) = \sqrt{K_{Sp_1}(z)\, K_{Sp_2}(z)} \tag{2.23}$$

Total flow is obtained integrating Darcy's equation over the saturated depth

$$Q_x = \int_{z_{inf}}^{z_{sup}} q_x(z)\, dz = -\int_{z_{inf}}^{z_{sup}} K_{eq}(z)\, J_x(z)\, dz \tag{2.24}$$

where

- z_{inf} = minimum $\left(\frac{N_{f1}}{cos(\alpha)} , \frac{N_{f2}}{cos(\alpha)} \right)$

- z_{sup} = maximum $\left(\frac{N_{t1}}{cos(\alpha)} , \frac{N_{t2}}{cos(\alpha)} \right)$

Applying the principle of superposition, lateral flow Q_x is given by

$$Q_x = \int_{z_{inf}}^{z_{sup}} K_{eq}(z) \frac{\psi_1(z) - \psi_2(z)}{\Delta x}\, dz$$

$$Q_x = \int_{\frac{N_{t1}}{\cos(\alpha)}}^{\frac{N_{f1}}{\cos(\alpha)}} K_{eq}(z) \frac{\psi_1(z)}{\Delta x}\, dz - \int_{\frac{N_{t1}}{\cos(\alpha)}}^{\frac{N_{f1}}{\cos(\alpha)}} K_{eq}(z) \frac{\psi_2(z)}{\Delta x}\, dz \tag{2.25}$$

The transfer of moisture between elements due to imbalances in pressure distribution can be computed as the difference between the flows that

would result considering pressure distribution in both pixels independently. That flow can be obtained substituting for the expressions of K_{eq} and and Ψ. The distribution of Ψ as a function of n is given by Equation (2.11), which can be readily expressed in terms of z. K_{eq} is expressed as a function of z as

$$K_{eq} = \sqrt{K_{0p1}\, K_{0p2}}\; e^{-f\, z \cos(\alpha)} \tag{2.26}$$

Substituting for Ψ and K_{eq} in the first term of Equation (2.25), carrying out the integration over z, and considering a section of width W, we obtain the expression for the moisture outflow from pixel 1, Q_{pout}, considering only pressure distribution in pixel 1

$$Q_{pout} = K_{0eq} \frac{W}{x_2 - x_1} (N_f - N_t) \left[\frac{N_t e^{fN_f} - N_f e^{fN_t}}{e^{fN_f} - e^{fN_t}} + \frac{1}{f} - \frac{N_f + N_t}{4} \right] \tag{2.27}$$

All variables in Equation 2.27 refer to values in pixel 1. The second term in Equation (2.25) corresponds to moisture inflow into pixel 1 considering only pressure distribution in pixel 2. The expression is analogous to Equation (2.27), substituting 1 by 2. The net outflow from pixel 1 is given by the difference between outflow and inflow, as shown in Equation (2.25).

2.2.4 Runoff generation

Two modes of runoff generation are represented in DBS: infiltration excess runoff and return flow. Both are estimated as final output of the equations described above for front movement and lateral moisture transfer. We first analyze the different runoff-generation states of the pixel, according to front position. Then, infiltration excess-runoff, which is a direct consequence of pixel state, is analyzed. The problem of return flow is considered last, since return flow is obtained as a result of global moisture balance in the soil column, including the infiltration input in addition to subsurface lateral inflows and outflows.

Pixel states

The final result of the one-dimensional model of infiltration is a description of the moisture profile along the soil column. Depending on the position of the top and the wetting fronts, the soil column can be in four distinct states, which have different runoff-generating potential. In order to study the runoff generation in the basin, four basic pixel situations should be considered, as represented in Figure 2.6

- *Unsaturated*: The wetting front is at some depth above the water table, but saturation has not been reached yet. The soil column generates purely infiltration excess runoff only. Its runoff generation potential is controlled by its surface hydraulic conductivity exclusively.

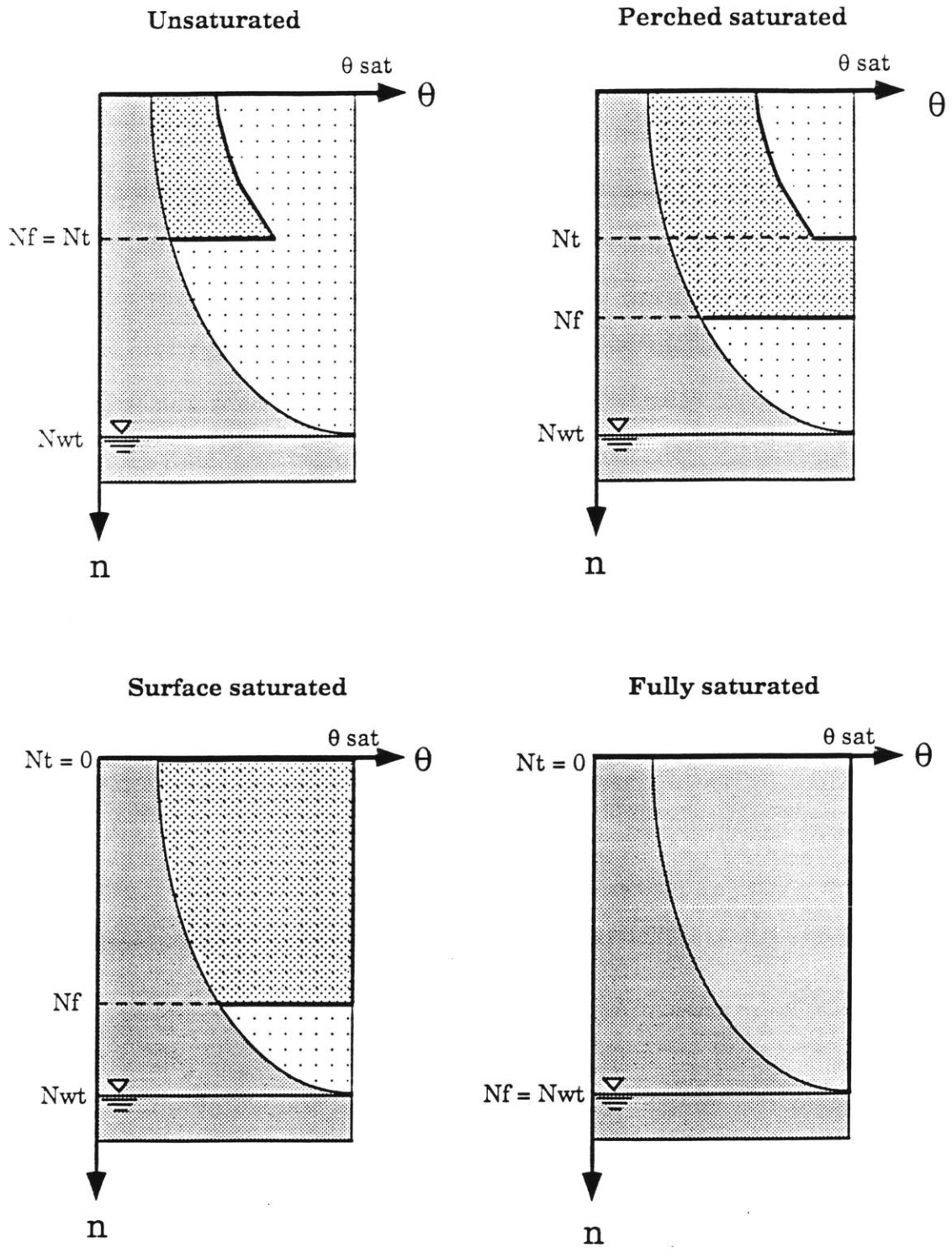


***Figure 2.6**: Representation of the four possible runoff-generation states of the pixel.*

- *Perched saturated*: The wetting front has reached the level of saturation N^*, and the top front is still at some depth below the surface. The soil column generates infiltration excess runoff. Its external behavior is similar to that of an unsaturated column, but, as the top front approaches the surface, the column becomes closer to saturation. An evaluation of how close the soil column is to being completely saturated is the water storage that still remains free in the pixel above the top front.

- *Surface saturated*: The wetting front is at some depth above the water table and the top front is at the surface. The runoff generation properties of the column are not function of its surface hydraulic conductivity, but of the harmonic mean hydraulic conductivity of the saturated depth. The column generates infiltration excess runoff, but the infiltration capacity is a function not only of the soil type, but also of the moisture state. In general, the infiltration capacity is significantly lower than that of the unsaturated case. The column may also generate return flow, if infiltration plus subsurface flows into the pixel exceed subsurface outflows plus the storage increment provided by the progression of the wetting front.

- *Fully saturated*: The wetting front has reached the water table and the top front is at the surface. The soil column behaves effectively as if the water table had reached the surface. The infiltration capacity of the pixel is equal to the inter-storm recharge rate R_i. No rainfall from the storm can infiltrate, and therefore, its tendency to generate surface runoff and return flow is maximum.

Infiltration excess runoff

The infiltration-excess runoff is a direct consequence of the infiltration capacity of the soil, which is in turn a function of front position. Actual infiltration is obtained by comparing the infiltration capacity of the soil with the rainfall rate. If the rainfall rate is higher than the infiltration capacity,actual infiltration equals the infiltration capacity and infiltration-excess runoff is generated. If we designate the infiltration capacity as I_{max}, three basic modes can be considered, comprising the four possible pixel states.

For unsaturated (N_f=N_t) and perched saturated ($0 < N_t < N_f$) pixels, the maximum infiltration capacity is only controlled by surface saturated conductivity

$$I_{max} = K_{0n} \cos(\alpha) \tag{2.28}$$

When the pixel is in the surface-saturated state ($N_t = 0$, $N_f < N_{wt}$), the infiltration capacity is abruptly reduced

$$I_{max} = K_{0n} \frac{f N_f}{e^{fN_f} - 1} \cos(\alpha) \tag{2.29}$$

After the top front reaches the surface of the soil, infiltration into the soil is no longer controlled by the infiltration capacity of the unsaturated upper layers (surface normal hydraulic conductivity K_{0n}), but by the

infiltration capacity of all the saturated zone as a whole. which extends from the surface down to the wetting front.

The fully-saturated state (N_t=0, N_f=N_{wt}) is the limiting case in which the saturated zone extends down to the water table. For these pixels, the infiltration capacity is in equilibrium with the inter-storm recharge rate, and storm rainfall cannot infiltrate,

$$I_{max} = 0 \tag{2.30}$$

The actual infiltration I is given in all cases by

$$\begin{aligned} I &= R \quad \text{if} \quad R \le I_{max} \\ I &= I_{max} \quad \text{if} \quad R > I_{max} \end{aligned} \tag{2.31}$$

Designating the the infiltration-excess runoff by R_i, it is given by

$$R_i = R - I \tag{2.32}$$

Return flow

Return flow is generated when the pixel is saturated, as a consequence of the moisture balance. The evolution of the total moisture content in the pixel is given by

$$\frac{dM_t}{dt} = \frac{dN_f}{dt}\,\theta(R_i,N_f) + I + \frac{Q_{lin} - Q_{lout}}{A} \tag{2.33}$$

The first term of Equation (2.33) corresponds to the moisture increment due to the vertical displacement of the wetting front. Since M_t is defined as moisture above the wetting front (see Equation 2.4a)), moisture accounting should include the increment due to the incorporation of the initial moisture distribution to our reference volume. The second term accounts for the infiltration, and is given by Equation (2.31). The last term includes lateral subsurface inflows and outflows from the pixel due to interactions with contiguous pixels, as described in Section 2.2.2, normalized by the horizontal area of the element, A.

The model assumes that all moisture inflows accumulate in the area of the soil column affected by the storm, that is, above the wetting front. Therefore, the total moisture content has an upper limit, set by $N_f\theta_s$, which corresponds to surface saturation. Whenever the sum of the previous moisture content in the element and the net moisture inflow exceeds the saturation volume above the wetting front, return flow is generated. If the pixel is not saturated at the surface (unsaturated or perched-saturated states), it has a reservoir above the top front to absorb subsurface moisture inputs, and it is unlikely to generate return flow. However, if the pixel reaches the surface-saturated state, the limit to the increase of moisture content is only set by the speed at which the wetting front progresses downwards, and the potential to generate return flow is very high. In this case, the maximum moisture increment allowed is

$$\left(\frac{dM_t}{dt}\right)_{max} = \frac{dN_f}{dt}\theta_s \qquad (2.34)$$

and therefore, return flow is generated whenever

$$I + \frac{Q_{lin} - Q_{lout}}{A} \geq \frac{dN_f}{dt}[\theta_s - \theta(R_i, N_f)] \qquad (2.35)$$

In that condition (surface-saturated pixel satisfying Equation 2.35), the return flow generated R_r is equal to

$$R_r = I + \frac{Q_{lin} - Q_{lout}}{A} - \frac{dN_f}{dt}[\theta_s - \theta(R_i, N_f)] \qquad (2.36)$$

In the limiting case of a fully-saturated pixel, $\frac{dN_f}{dt} = 0$, and the potential for generating return flow is maximum.

Return flow is clearly an effect of lateral flows. In isotropic terrain, lateral transport of water is usually small compared to vertical transport for a given pixel. Rainfall effects are confined to the area above the wetting front, which is the depth affected by the storm. For a typical storm, wetting front penetrations (l) are of the order of a few meters at the most, while pixel sizes (L) are tens or hundreds of meters. In isotropic terrain, both lateral and vertical flows are of the same order of magnitude, and hence the moisture inflow into a pixel due to rainfall infiltration is one or two orders of magnitude larger than either the inflow or the outflow due to lateral flow, because vertical flow acts on LxL and lateral flow acts on lxL. Furthermore, the model assumes that inter-storm lateral subsurface flows are in equilibrium with the average recharge rate, and it only considers the perturbations introduced in the equilibrium lateral flow by a given storm, which are small. Since it is the net difference between inflow Q_{in} and outflow Q_{out} what is of concern, the differences in the order of magnitude between net lateral input and vertical input are even greater.

Therefore, unless pixel sizes are of the order of wetting front penetrations, non-convergent lateral flows in isotropic terrain could be neglected while computing the net moisture balance in the pixel. However, if the soil is highly anisotropic, lateral transport of water is significant, because lateral flows are considerably larger than vertical flows and can compensate for the difference in the cross-sectional areas upon which they operate. Moreover, if the pixel is saturated, lateral flow becomes extremely important, because it becomes return flow directly, which is added to surface runoff.

The total runoff generated by the pixel is designated by R_f, and it is the sum of the infiltration excess runoff R_i, given by Equation (2.32), and the return flow R_r, given by Equation (2.36) or mass balance considerations in a discretized scheme

$$R_f = R_i + R_r \tag{2.37}$$

2.3 Surface flow routing

The second aspect in the computation of basin response is the routing of the runoff along the stream channels down to the basin outlet. Algorithms for extracting the river channel network from a DEM can be used used to identify the path that runoff generated at every grid point would follow down to the basin outlet (Tarboton et al., 1989), which can provide us with a physical basis to estimate the distributed basin response.

2.3.1 Basic routing equation

The most general formulation of the distributed response of a basin is the well-known distributed convolution equation

$$Q(t) = \int_A \int_0^t R_f(x,y,\tau)\, h(x,y,t-\tau)\, d\tau\, dA \qquad (2.38)$$

where $Q(t)$ is the resulting hydrograph at the outlet, $R_f(x,y,\tau)$ is a function describing the distribution of runoff rate generation per unit area and $h(x,y,t)$ is the instantaneous response function of the element of area dA located at coordinates (x,y). For a linear system, the instantaneous response function is constant during the event. For a non-linear system, the instantaneous response function may vary during the event, according to the varying transport conditions in the drainage network.

Two physical processes control the characteristics of the instantaneous response function: convection and dispersion. Convection refers to the bulk transport of the elementary response through the drainage network down to the basin outlet. Dispersion refers to the spreading experimented by the elementary response during the transportation, due mostly to the non-uniform velocity distribution across the channel cross section. Considering the elementary runoff generation a Dirac delta function at some time t_o, $\delta(t_o)$, the instantaneous response function due exclusively to convective transport would be another delta function, lagged by an interval t, $\delta(t_o+t)$. t is the time of travel of the response from the element dA to the basin outlet. The effect of dispersion would be to spread the response around the mean value t_o+t. Different

functional forms could be adopted to account for dispersion (Gaussian, gamma function, etc). Tracer experiments in natural streams (Pilgrim, 1979) show the relative importance of both phenomena. Although there is a considerable amount of dispersion, convection is the predominant phenomenon in midsize catchments.

The detailed geomorphological description provided by DEM's provides a good estimate of the length of the travel path. The evaluation of the time of travel along that path is a more complicated problem, since it involves the estimation of flow velocities. Many studies have been carried out in natural streams, measuring average flow velocities in different locations in a basin for different flow conditions (Leopold and Maddock, 1953; Askew, 1970; Pilgrim, 1979; Beven et al., 1979), suggesting that annually averaged velocities tend to remain constant along a given river system. However, for a given episode, temporal flow variations can be expected, according to the transient conditions of the rising and falling limbs of the hydrograph.

For a certain event, flow equations could theoretically be solved for both overland and stream flows, given that there is enough knowledge of the geometry and of the hydraulic characteristics of the channels. An analysis of the hydraulic conditions of the flow clearly suggests that the linearity assumption of constant velocity along the drainage network for all times is not valid. Flow conditions change significantly during the storm, and that influences flow velocities and times of travel throughout the basin. Although attempts have been made at solving simplified forms of the flow equations in distributed models (Huggins and Monke, 1968; James and Kim, 1990), the approach has always proven to be too demanding in terms of data and computational requirements for

application to midsize basins (of the order of hundreds of square kilometers). Our problem here is to find a parameterization of the problem that be simple enough to operate efficiently in a distributed model, but that can capture the most relevant features observed in the basin response. It is an important requirement to keep the number of model parameters to a minimum, in order to be able to obtain a parsimonious calibration.

2.3.2 Distributed instantaneous response function

The objective of the solution adopted for the instantaneous response function is to differentiate between the two transport mechanisms that operate in overland and stream flow, and to take into account some form of nonlinearity in basin response. Convection is assumed to be the dominant factor in basin response, and, although it is acknowledged that dispersion can also be of importance, its effect is neglected to a first approximation, in order to keep model parameters to a minimum. Therefore, the distributed instantaneous response function is assumed to be a Dirac delta function, with a delay equal to the time of travel from the location of the element to the outlet of the basin. Inclusion of dispersion effects in the formulation would be straightforward, substituting the delta function for other parameterization of the instantaneous response function. The travel path l_T corresponding to a typical hillslope element consists of a hillslope fraction l_h and a stream fraction l_s

$$l_T = l_h + l_s \tag{2.39}$$

Overland flow is assumed to be a quickly channelized process, and therefore the model does not account for infiltration in unsaturated elements during the routing process. The travel time is defined according to the assumption of uniform velocities for both overland and stream flows throughout the basin for a given time, although travel velocities are allowed to vary as the storm progresses, accounting for the changing flow conditions in the streams. If $v_h(\tau)$ is the hillslope velocity and $v_s(\tau)$ is the stream velocity at time τ, the travel time t_τ for the typical hillslope element is given by

$$t_\tau = \frac{l_h}{v_h(\tau)} + \frac{l_s}{v_s(\tau)} \tag{2.40}$$

The uniform surface velocity approximation allows for a simple computation of the hydrograph at the outlet of the basin. The instantaneous response function of the basin element located at (x,y) at time τ would be the Dirac delta function given by

$$h_\tau(x,y,t) = \delta\left(\frac{l_h(x,y)}{v_h(\tau)} + \frac{l_s(x,y)}{v_s(\tau)}\right) \tag{2.41}$$

An incremental basin response is estimated independently for every time step τ routing the runoff generated at every pixel

$$q_\tau(t) = \sum_{(x,y)\,\in\,\text{Basin}} R_{f\tau}(x,y)\, h_\tau(x,y,t)\, \Delta x\, \Delta y \tag{2.42}$$

where $R_{f\tau}(x,y)$ is the runoff rate generated in the element at location (x,y) at time τ and $\Delta x \Delta y$ is the area of the element. The total basin response at time T is obtained adding the incremental responses since the beginning the storm

$$Q(t) = \sum_{\tau=0}^{\tau=T} q_{\tau}(t) \qquad (2.43)$$

This methodology allows the estimation of basin response once an estimate for hillslope and stream velocities is available at every time step.

2.3.3 Non-linearities in basin response

Studies and experiments show evidence of non-linear behavior in basin response. Several expressions for the average velocity or time of travel have been proposed in the literature. The relationship between discharge and velocity during a storm event has been studied using three approaches: the lag time, the tracer experiments and the rating curve.

A significant fraction of the work done deals with the evaluation of the lag time at basin scale , which can be defined as the difference between the center of mass of excess rainfall to the center of mass of the resulting runoff. The lag time lumps into one single parameter all the spatial and temporal variability of basin response, and it is therefore a good estimate of the overall non-linearity of basin response. Lag time was found to be strongly correlated to the magnitude of discharge for high flows. Several parameterizations have been proposed relating lag time to

some characterization of discharge. Askew, (1970) proposed a power law of the form

$$t_L = a\, q_w^{\ b} \tag{2.44}$$

where q_w is a weighted mean discharge. He found exponents ranging from -0.2 to -0.32, with reasonably good correlation coefficients. Other authors (Laurenson, 1964; Minshall, 1960) fitted laws of similar form considering peak or mean discharge with satisfactory results. Boyd et al. (1979) verified that the power relationship proposed for average discharge values could be extended to represent instantaneous values in a non-linear storage routing equation.

Another group of analyses deals with the study of tracer experiments, measuring the time of travel and dispersion of tracer injections on stream reaches of small watersheds. These experiments offer information on flow velocities along a given path within the basin, but results are averaged in time, and usually only cover a small range of flows for a given location. These studies (Beven et al., 1979; Pilgrim, 1977; Kellerhals, 1970) show that average velocities increase nonlinearly both downstream and with increasing discharge, although an upper bound was found for velocities in high flows. Power laws of the form

$$v_r = m\, Q^n \tag{2.45}$$

have also been fitted to these cases. Here v_r is the average velocity along a reach and Q is a measure of discharge, usually average discharge during the experiment, although Pilgrim (1977) found better correlations using

the peak discharge of the event. The variability of the exponent n is larger than that of the exponent b in the case of lag time, with an average value around 0.5, but with results ranging from 0.2 to 0.8.

A third case is the study of the theoretical or empirical relationships between velocity and discharge at a given location for different flow regimes, what is known as the rating curve. That relationship is highly dependent on the local slope and on the shape and roughness of the cross section of the channel, and it therefore does not have a general validity. Mein et al. (1974) assumed uniform flow governed by Manning's resistance formula and obtained exponents for the velocity-discharge relationship ranging from 0.25 to 0.4 for different cross-section shapes. They also cite several field studies in gaging stations, selecting an average value of 0.29 for their runoff routing method. Beven (1979) suggested that for high flows the power law usually assumed in kinematic routing models tends to overpredict flow velocities, and proposed a modified method that considers constant velocities for high flows.

None of these three approaches coincide with our problem of estimating average flow velocities along a certain path to estimate the time of travel for a given instant during a storm. Nevertheless, all of them suggest a that a power relation between discharge and flow velocities might hold, at least on a statistical sense. The solution adopted is based on a relationship of that kind, and tries to capture the fact that travel times are a function of the amount of water present in the basin. The value selected to estimate flow velocities is the discharge at the outlet of the basin at the same time. Basin discharge provides a rough estimate of the conditions in the drainage network, and has the computational advantage

of being readily available at all times. Therefore, the following equation is applied to obtain average channel velocities in the basin

$$v_s(\tau) = c_v[Q(\tau)]^r \tag{2.46}$$

v_s is the channel velocity at time t, $Q(t)$ is the discharge at the outlet of the basin at time t, and c_v and r are coefficients. Since no direct measurements are available, the coefficients have to be estimated through calibration. It is also considered that the ratio of stream velocity to hillslope velocity is a constant, K_v. The travel velocity in hillslopes is given by

$$v_h(\tau) = \frac{v_s(\tau)}{K_v} \tag{2.47}$$

where K_v is the ration of stream velocity to hillslope velocity. K_v also has to be estimated through calibration.

This scheme allows for the introduction of some non-linearities in basin response. It should be interpreted only as a first order approximation to deal with basins that exhibit a clearly non-linear behavior.

CHAPTER 3

Model Performance

The performance of DBS is analyzed in this chapter. The computer implementation of DBS is discussed in Chapters 4 and 5. Here the computer implementation is used to assess model behavior and to evaluate model performance. The presentation of model performance is divided into two sections. The first section contains a sensitivity analysis whose goal is to identify the influence of model parameters in model performance. The second section is an example of the practical use of the model. The model is calibrated with data for the Sieve river basin, and the quality of the calibration is verified with a different data set.

3.1 Sensitivity analyses

The sensitivity analyses presented in this section evaluate the influence of model parameters in the final result and assess the capability of the distributed basin model to reproduce the physical mechanisms that it is intended to simulate.

Although the final goal of the model is to simulate the global response of the basin to a given storm, the fact that it is physically-based and distributed implies that the model should acceptably reproduce the internal behavior assumed for the basin. It means that, from the modeling

point of view, it is a requisite that the model be able to reproduce not only the total hydrograph for a given storm, but also all the internal runoff generation mechanisms postulated in the model definition. The goal of the sensitivity analysis is therefore twofold:

- Explore model sensitivity to the different parameters defined, identifying those responsible for a greater part of the response.
- Explore under which combinations of parameters the internal results produced by the model reproduce its expected behavior. Unfortunately, field data corresponding to internal behavior of a basin are usually not available, and thus model results can only be checked against working hypotheses about how the real watershed behaves.

The parameters that need investigation in the sensitivity analysis can be inferred from a qualitative picture of how the model works. DBS is a distributed, event-oriented model, intended to work for flood forecasting. Almost all low-frequency variations and slow processes are neglected. Modeling strategy consists basically in identifying the area of the basin which is saturated and describing its time evolution as a function of rainfall. The model reproduces the pattern of saturation areas in the watershed based mainly on topographical information. It is postulated that the saturated area develops either near the streams, in areas of convergence of subsurface flow, or in areas where local infiltration has been high. The good behavior of the model depends on the adequate mapping of the saturated area and on the correct representation of its evolution in time, as the storm progresses. Therefore, the key aspect to test in the sensitivity analyses is the evolution in time of the state of the basin, which is fundamental to understand how runoff is generated.

Sensitivity analyses were carried out on two different basin models: a simplified hillslope model and a real basin model. The hillslope model represents an idealized hillslope of limited height and infinite width, since no interactions are defined in the traversal direction. Different pixel size definitions and soil type distributions were considered. The real basin model was constructed from the digital elevation data available for the Sieve catchment, part of the Arno basin in Italy. Section 3.1.1 presents the results obtained with the hillslope model, and Section 3.1.2 presents the results obtained with the basin-scale model.

3.1.1 Hillslope model

An idealized model of a uniform slope was defined to study the behavior of the model in a system of simple geometry, in order to compare it with other analytical or simplified numerical solutions. A hillslope of constant slope was defined. The geometry of the hillslope is represented in Figure 3.1. The slope forms an angle of 10° with the horizontal datum and has a total length of 4000 m. The initial water table position follows a straight line. Water table depth is 10 m at the divide and 0 m at the bottom.

The objective of the hillslope sensitivity analysis is to evaluate the importance of the subsurface flow and of the interactions between pixels. The hillslope sensitivity analysis tests the validity of the approximate expressions proposed to model subsurface flow of moisture between contiguous pixels. It also studies how an array of pixels can model the behavior of a real hillslope, capturing the relevant aspects of the two-dimensional subsurface moisture flows. Qualitative and quantitative studies of subsurface moisture flow are available in the literature

(Troendle, 1985; Beven, 1981, 1982b), and therefore this simplified model is a good test case to compare model performance with the more elaborate solutions available for this problem.

***Table 3.1** Soil characteristics in the hillslope models*

Soil Type	K_{0n}	θ_s	θ_r	ε
Soil 1	0.25	0.48	0.09	7.5
Soil 2	5.1	0.49	0.109	3.6
Soil 3	16.6	0.52	0.064	3.5
Soil 4	45.0	0.56	0.109	3.6

The hillslope is represented by an array of consecutive pixels. Several models with different pixel sizes were defined, in order to test how pixel size affects the flow exchange between pixels or to what degree the model

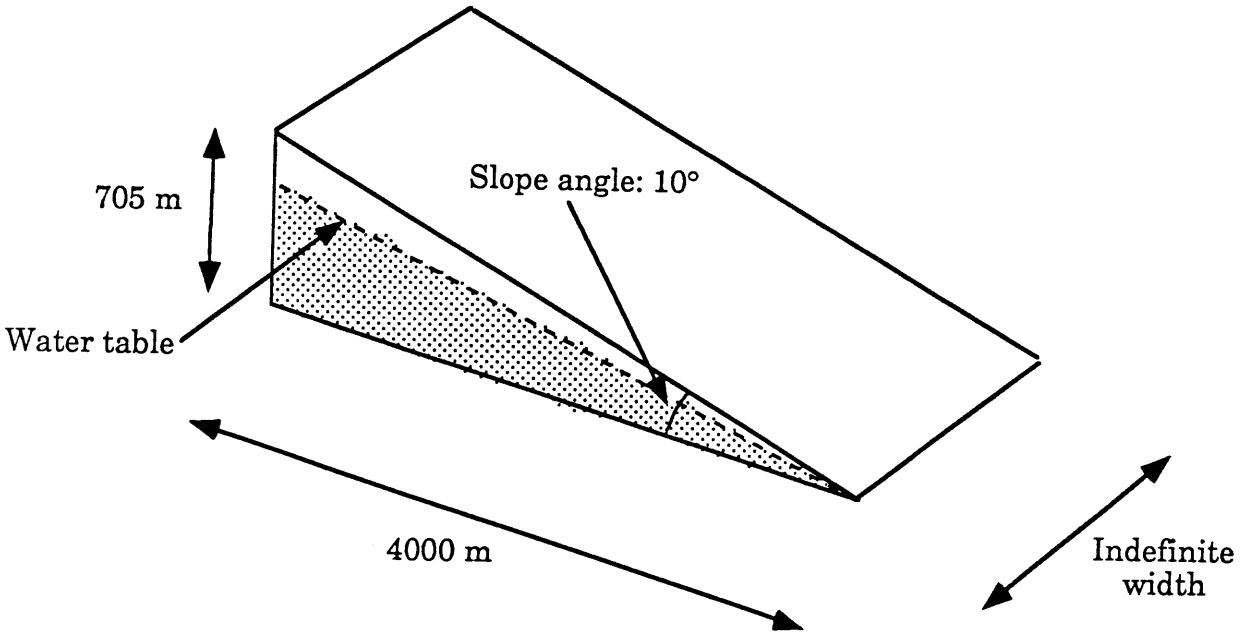
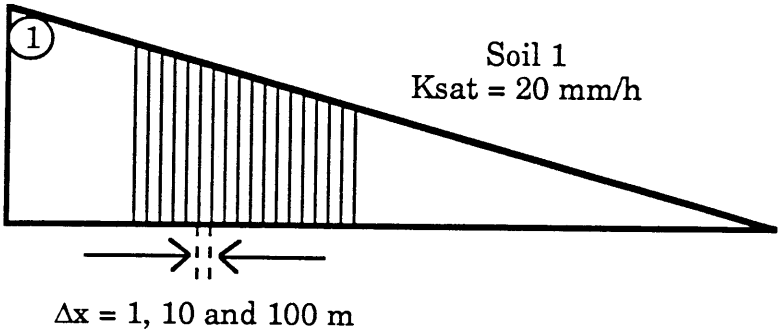


***Figure 3.1**: Schematic representation of the geometry of the hillslope used in the sensitivity analysis.*

reproduces in two dimensions the ideal situation assumed in the one-dimensional model of infiltration. Models with pixels of 1, 10 and 100 m were defined. Two situations were considered: hillslope formed by homogeneous terrain and hillslope formed by terrain of different soil types. Soil characteristics are summarized in Table 3.1 and hillslope configurations are represented in Figure 3.2. A standard parameter set, represented in Table 3.2, was used in the computations, where only the anisotropy ratio was varied, in the range 1, 10 and 100. The model was run with a constant rainfall intensity of 5.0 mmh^{-1}.

Model a



Model b

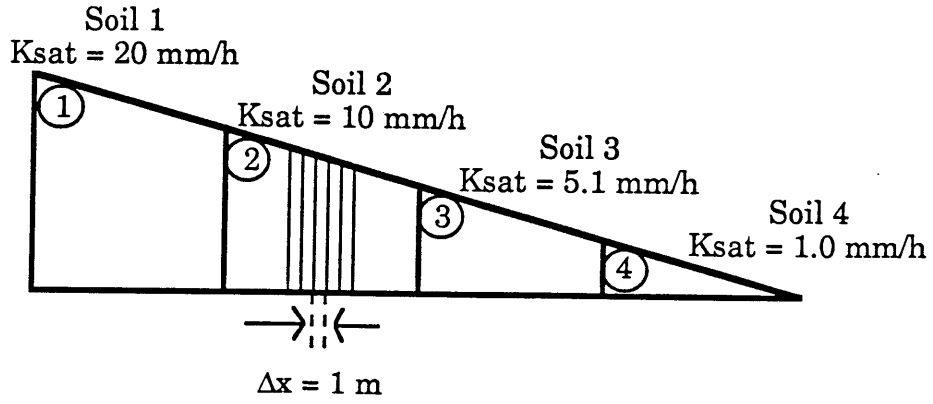


***Figure 3.2**: Hillslope models used in the sensitivity analysis.*

Two types of sensitivity analyses were carried out in the hillslope model. In the first set of sensitivity runs the effect of changing different computational features was analyzed, in order to set basic requirements to pixel size and computation time step. In the second set, the hillslope model was used to analyze how the model captures the two dimensional features of subsurface flow in the hillslope, accounting for the effect of soils of different characteristics with different anisotropy ratios.

***Table 3.2** Model parameters for the hillslope models*

Parameter	**Value**
Parameter f (mm^{-1})	1×10^{-4}
Anisotropy ratio a_r	1, 10, 100
Initial recharge rate (mmh^{-1})	0.01

Sensitivity to computational features

We test model sensitivity to pixel size and computation time step. The focus is on front position, since other variables of the pixel depend on the basic state variables. Front position is analyzed after 30 and 60 hours of uniform rainfall intensity over a hillslope with uniform soil type. A soil of 20 mmh^{-1} of saturated hydraulic conductivity at the surface and anisotropy ratio of 10 was selected. The case with the smallest pixel size and computation time step is taken as the base result.

Figure 3.3 shows terrain surface and front positions after 30 and 60 hours of simulation for a pixel size of 1 m. The water table level is taken as the reference level, and appears horizontal in the figure. In the upstream portion of the hillslope, the wetting and the top fronts coincide for both

cases, because saturation has not developed. As we move further downslope, the wetting front penetrates deeper, as a result of moisture migration downslope carried by the subsurface flow. A small area of perched saturation can be observed in the lower third of the slope, where the lines representing the wetting and the top front diverge. A sudden increase in the thickness of the saturated area is noticed when the wetting front (lower line) reaches the water table. The first pixel in that situation along the profile becomes fully saturated, since it receives all the subsurface flow coming from the upstream pixels. The saturation of the pixel appears as a spike in the diagram, and identifies the point of generation of return flow. Shortly after that, the top front (upper line) reaches the surface, and an area of saturation develops in the lower part of the hillslope, where the whole depth of the soil column is saturated.

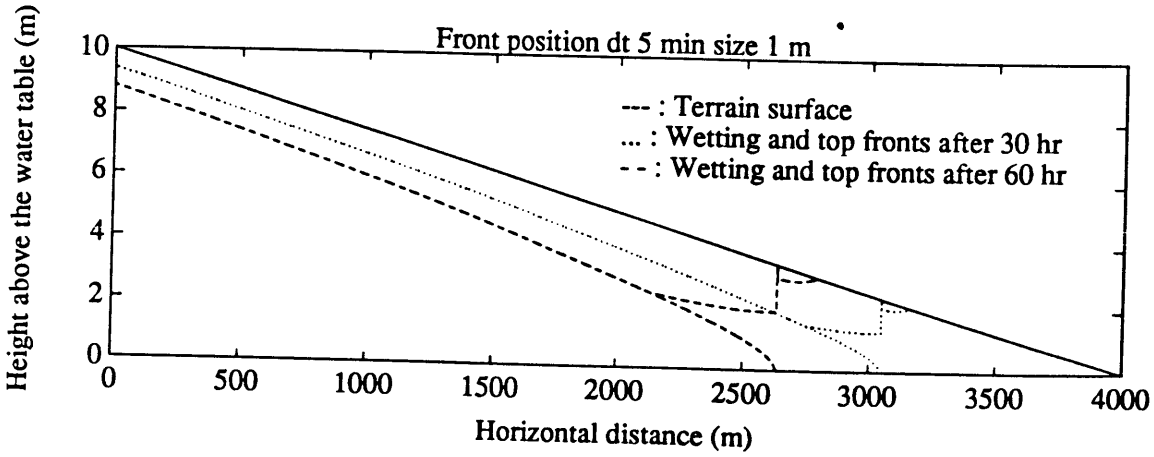


***Figure 3.3**: Terrain surface and front positions with respect to the water table level after 30 and 60 hours of simulation. The results correspond to hillslope model a. The pixel size is 1 m and the computation time step is 5 min.*

The saturated area generates all the runoff from the hillslope. The infiltration capacity of the hillslope is equal to the surface saturated hydraulic conductivity in the unsaturated and perched saturated areas, and is zero in the lower part of the hillslope where the whole depth of the soil column is saturated. The normal saturated hydraulic conductivity of the soil at the surface is 20 mmh^{-1}, and rainfall intensity is 5 mmh^{-1}. The model shows how downhill subsurface flow accumulates in the areas of shallow water table, generating runoff in a soil of surface infiltration capacity four times greater than rainfall intensity.

- *Pixel size*: The effect of considering increasing pixel sizes was analyzed, to evaluate its influence on numerical accuracy. Three different pixel sizes were considered: 1, 10 and 100 m. Figures 3.4 and 3.5 compare the results obtained with different pixel sizes for three computation time steps. Figure 3.4 presents the position of the fronts 30 hours after the initiation of rainfall, and Figure 3.5 presents the position of the front 60 hours after the initiation of rainfall. Each figure includes three plots, corresponding to computation time steps of 5 min, 30 min and 60 min respectively. Each plot compares model results for the three pixel sizes considered, and focuses on the lower part of the hillslope because in the upper part (unsaturated area) front positions for all pixel sizes are almost coincident. The figures show that all pixel sizes give similar results except in the area where the wetting front reaches the water table and the top front depth decreases sharply. In all cases, the full saturation of the first pixel whose wetting front reaches the water table is predicted, but the location of the pixel varies slightly from one case to another. This minor difference is due to the lack of spatial

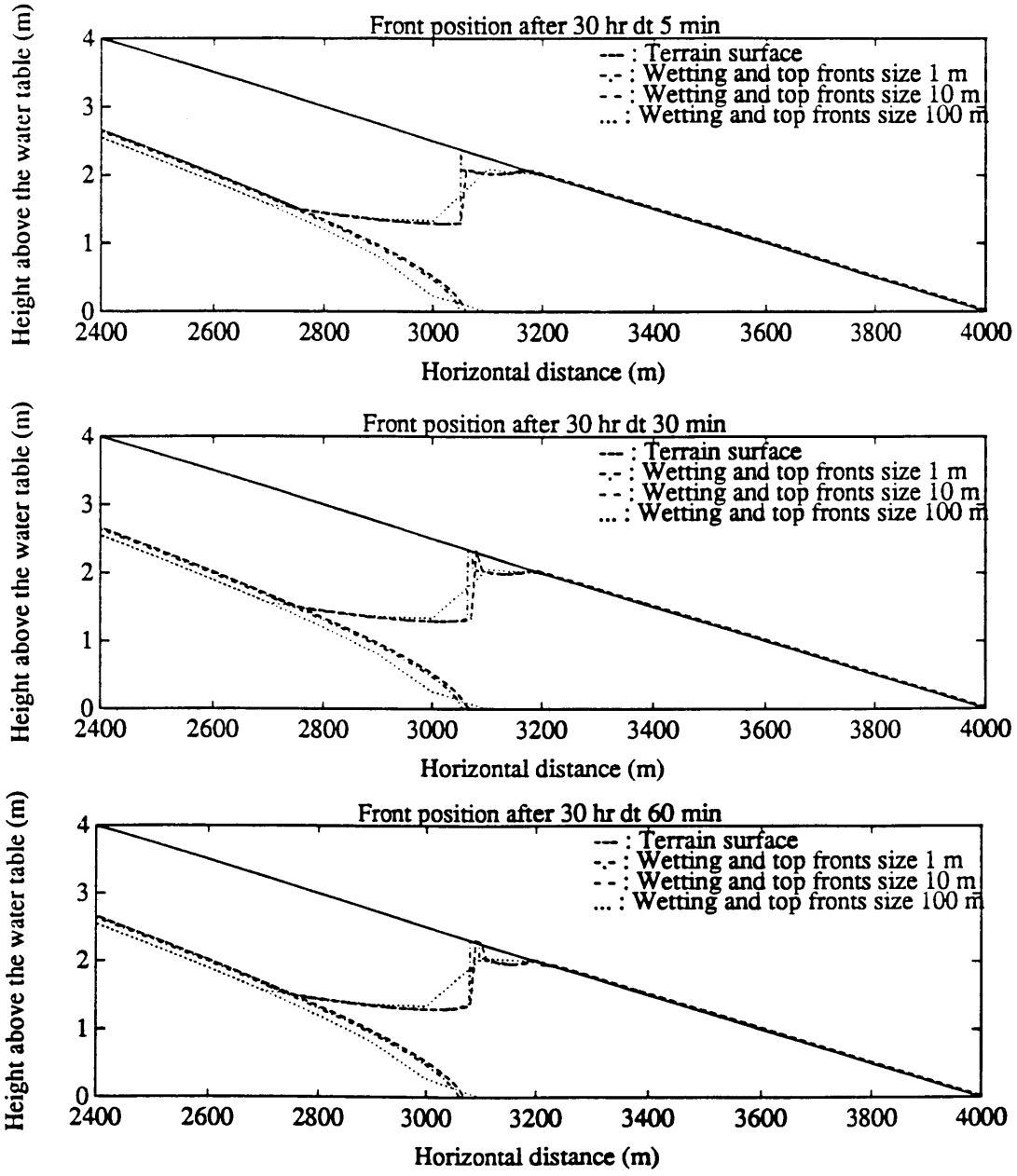


***Figure 3.4**: Sensitivity of front position to pixel size for computation time steps of 5, 30 and 60 min after 30 hours of simulation. The results correspond to hillslope model a.*

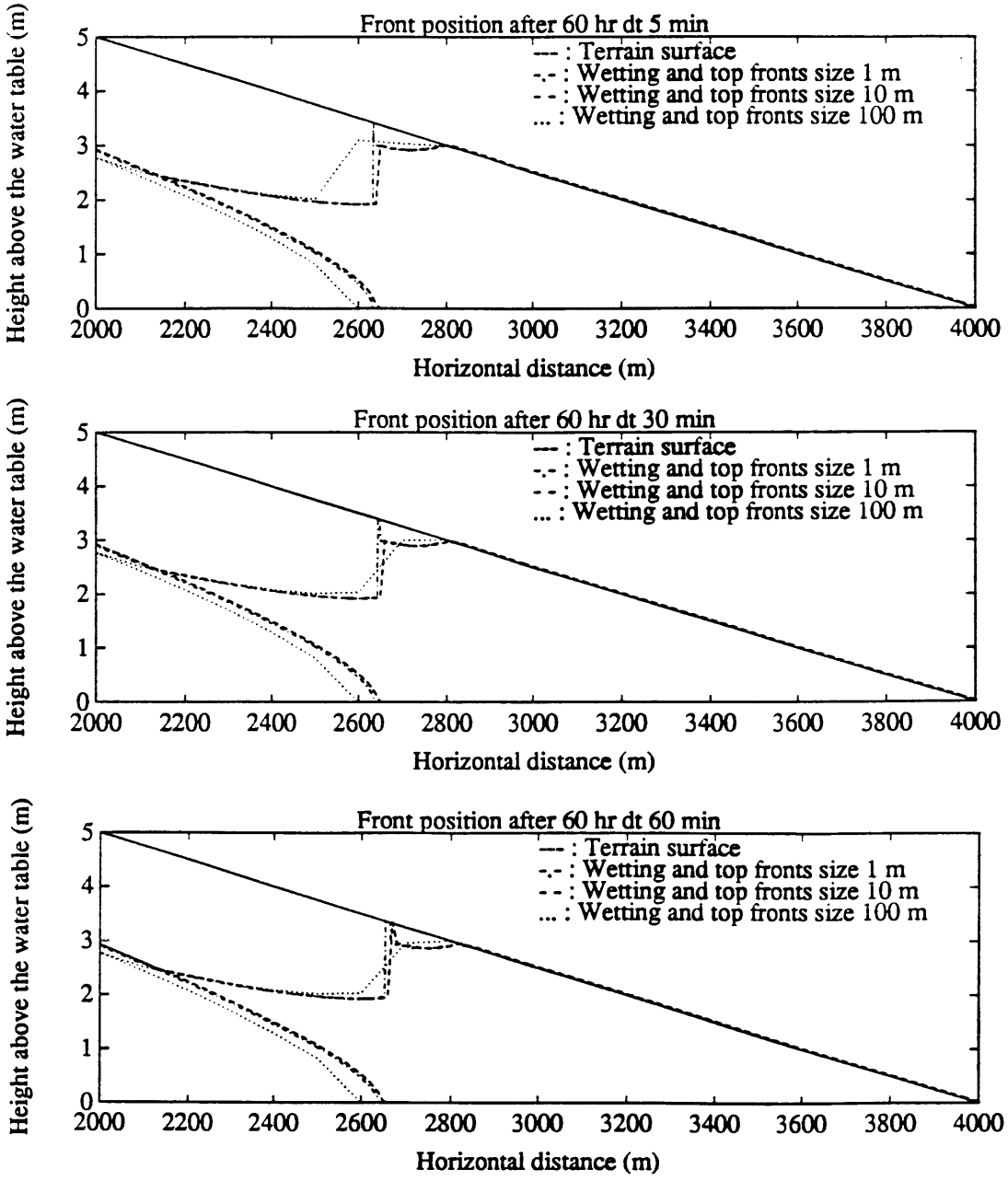


***Figure 3.5**: Sensitivity of front position to pixel size for computation time steps of 5, 30 and 60 min after 60 hours of simulation. The results correspond to hillslope model a.*

resolution in the model to identify the exact point at which this event occurs. In general, model behavior for different pixel sizes is very good.

It can be concluded that, at least for the uniform conditions considered in these sensitivity experiments, the errors resulting from considering a large pixel size (of the order of hundreds of meters) are not important. It was also verified (in the case of maximum definition: pixel size of 1 m) that the transient effects of the boundaries both upslope and downslope only extend a few meters, because inflow and outflow for a given soil column are quickly balanced. This result agrees with the observations of Philip (1991a).

The impact of pixel size is very much a function of the magnitude of lateral flow. Lateral transfer of moisture between pixels is only important when there is an unbalance in the conditions upstream and downstream of the soil column. That happens when there is high variability of soil type or when there is the possibility of generating return flow. For uniform soil type and deep water table, lateral flow is not important. Subsurface interactions between pixels only take place through the saturated area. In most cases, the saturated area extends only up to a few meters of depth (1-2 m), and, unless soil types in the neighboring pixels are very different, the net balance of the amount of moisture advected by lateral flow is negligible compared to the infiltration. It means that the one-dimensional vertical effect (common to all pixel sizes) is dominant when lateral flows are small.

Pixel size will therefore be conditioned more by the variability of slopes and soil types in the basin and by the validity of the ideal conditions formulated for the one-dimensional model than by numerical accuracy constraints. As long as the physical variability in the real basin allows for an accurate description using large pixel sizes, the numerical errors are

not significant. However, there is a limit to pixel size imposed by the physical interpretation of model variables. For pixels of hundreds of meters to the side, the assumption of a wetting front whose surface is parallel to the terrain surface is unrealistic. For these pixel scales, the model applied to the subgrid is more a conceptual representation rather than a physically-based representation.

- *Computation time step*: Figures 3.6 and 3.7 present the results of the sensitivity analyses to computation time step. Three different computation time steps were considered: 5 min, 30 min and 60 min. Figure 3.6 presents front positions after 30 hours of simulation and Figure 3.7 presents front positions after 60 hours of simulation. Three plots, corresponding to pixel sizes of 1, 10 and 100 m, are presented in each figure. In each plot, front positions in the lower part of the hillslope are compared for the different computation time steps. The results are very good for the time step of 30 min, which almost coincides with the time step of 5 min. Only minor differences are observed for the time step of 60 min, except in the case of pixel size of 100 m after 60 hr, where the sudden increase of the top front position is displaced one grid element.

Since the model is largely governed by the mass conservation equation, the effect of the computation time step is only of second order. The time evolution of front velocities given by Equations (2.16) and (2.17) is very smooth for uniform conditions. The changes in front velocities are very small, and therefore, the effect of considering a constant value even for a long period of time is largely irrelevant. The time increment of the simulation is more constrained by input variability than by computational requirements. Nevertheless, it should be noted that, since an explicit

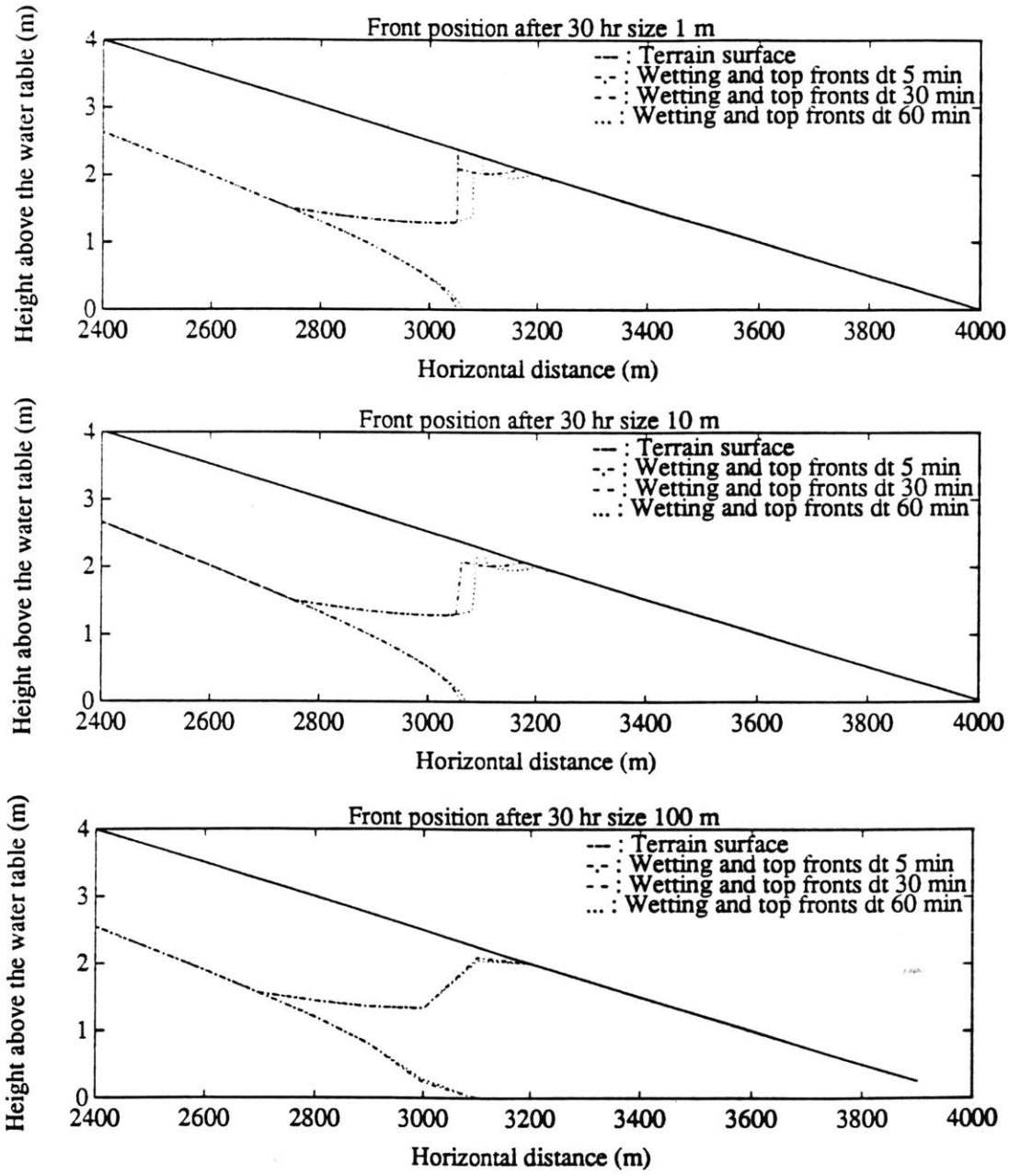


***Figure 3.6**: Sensitivity of front position to computation time step for pixel size of 1, 10 and 100 m after 30 hours of simulation. The results correspond to hillslope model a.*

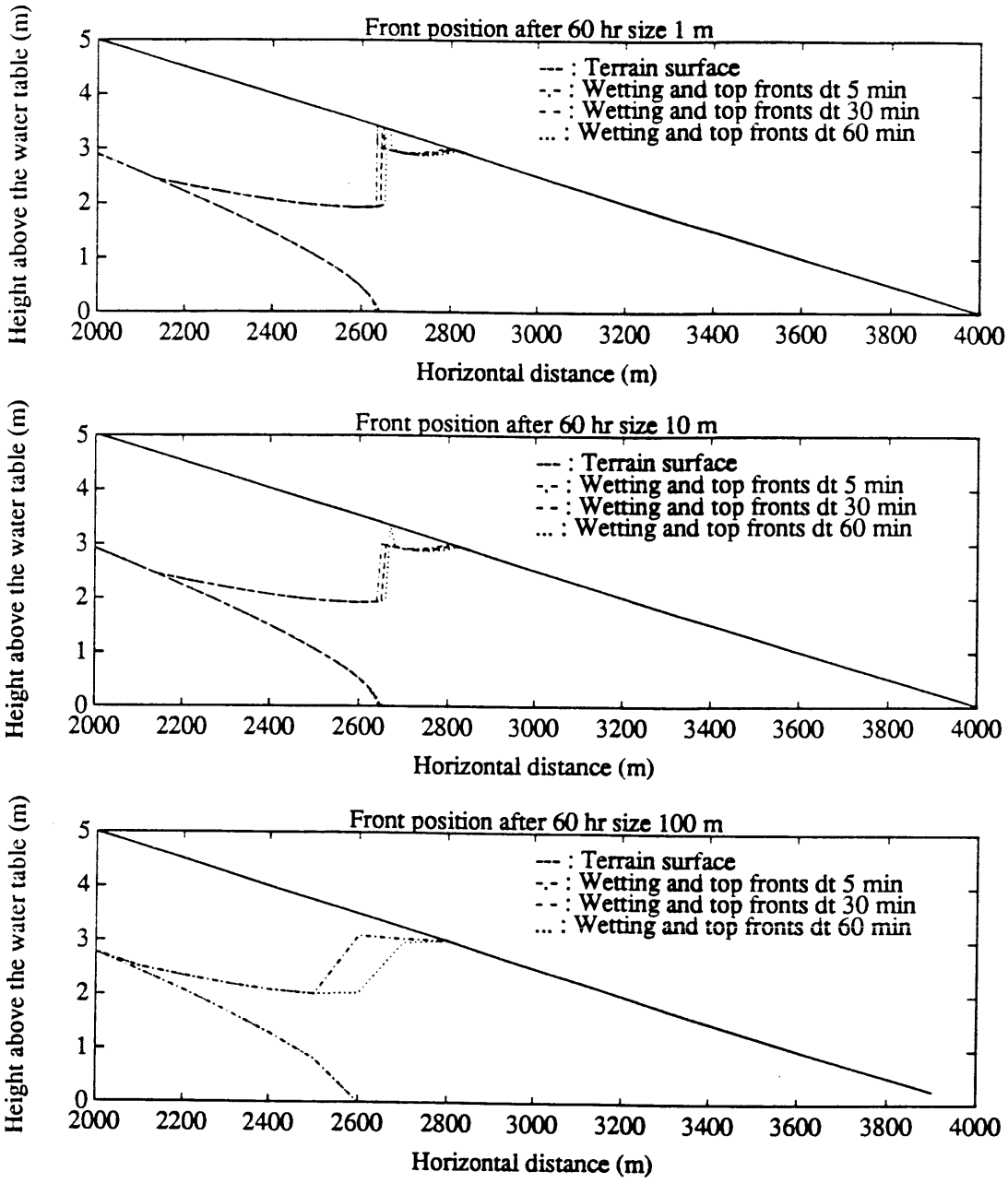


***Figure 3.7**: Sensitivity of front position to computation time step for pixel size of 1, 10 and 100 m after 60 hours of simulation. The results correspond to hillslope model a.*

numerical scheme was adopted, the sensitivity to changes in the conditions could be very high. The results presented here are always well-behaved because the sensitivity analyses were made on a hillslope of homogeneous terrain under uniform rainfall intensity. Practical cases present much stronger temporal and spatial variability where there is more potential for numerical problems. Therefore, extreme care should be taken in guaranteeing numerical stability.

There are also situations in which the computation time step is critical. If soil anisotropy is high, the ratio of lateral to vertical flow is also high in the same proportion. For very narrow pixel sizes (of the order of a few meters), the net volume of water entering the soil column laterally is many times larger that that entering the column vertically. Model equations are formulated mainly for vertical flows, and lateral flows are only envisioned as correction terms. For small pixels in anisotropic terrain, lateral flow is completely dominant, and the numerical formulation of the model is inappropriate. In these cases, the computation time step must be reduced in order to prevent numerical instabilities caused by the fact that front velocity is not a consequence of the gravity-driven, one-dimensional infiltration of water, as assumed by the model, but a result of of the interaction of both vertical and lateral flow. The instabilities are caused by the explicit numerical scheme, and usually lead to oscillating front positions in consecutive pixels, alternating in successive time steps. These instabilities are eliminated either defining a smaller time step or considering larger pixels, where the difference between lateral and vertical flow is compensated by the cross-sectional areas considered.

As a concluding remark, it should be noted that the conditions in this idealized hillslope are very special, and the results obtained in this sensitivity analysis cannot be extrapolated to real basins without careful consideration of the differences. This analysis has dealt only with uniform soil conditions, in elements connected in a very simple linear structure. Irregular distribution of soil types and flow convergence are usually found in natural basins. They are factors that can modify the conclusions of the sensitivity analysis and should therefore be taken into account.

Sensitivity to physical features

We test model sensitivity to two basic physical features: soil type and anisotropy ratio. The objective in this sensitivity analysis is to explore how the runoff generation mechanisms of the model work for very simple cases, which can be compared to other theoretical hillslope models. Two features are analyzed: front position and runoff generation along the hillslope. The hillslope model used is represented as *model b* in Figure 3.2. Four bands of equal length with different soil properties were considered. A time step of 30 min and a pixel size of 1 m were used in the computations.

Figures 3.8 to 3.10 present the results obtained for the three anisotropy ratios considered. Figure 3.8 shows results for a_r=1, Figure 3.9 shows results for a_r=10 and Figure 3.10 shows results for a_r=100. Each figure presents results 30 hours and 60 hours after the beginning of rainfall. For each case, the upper plot shows front positions and the lower plot shows runoff rate distribution along the hillslope. The four basic runoff generation mechanisms can be observed in every plot. Pixels in the

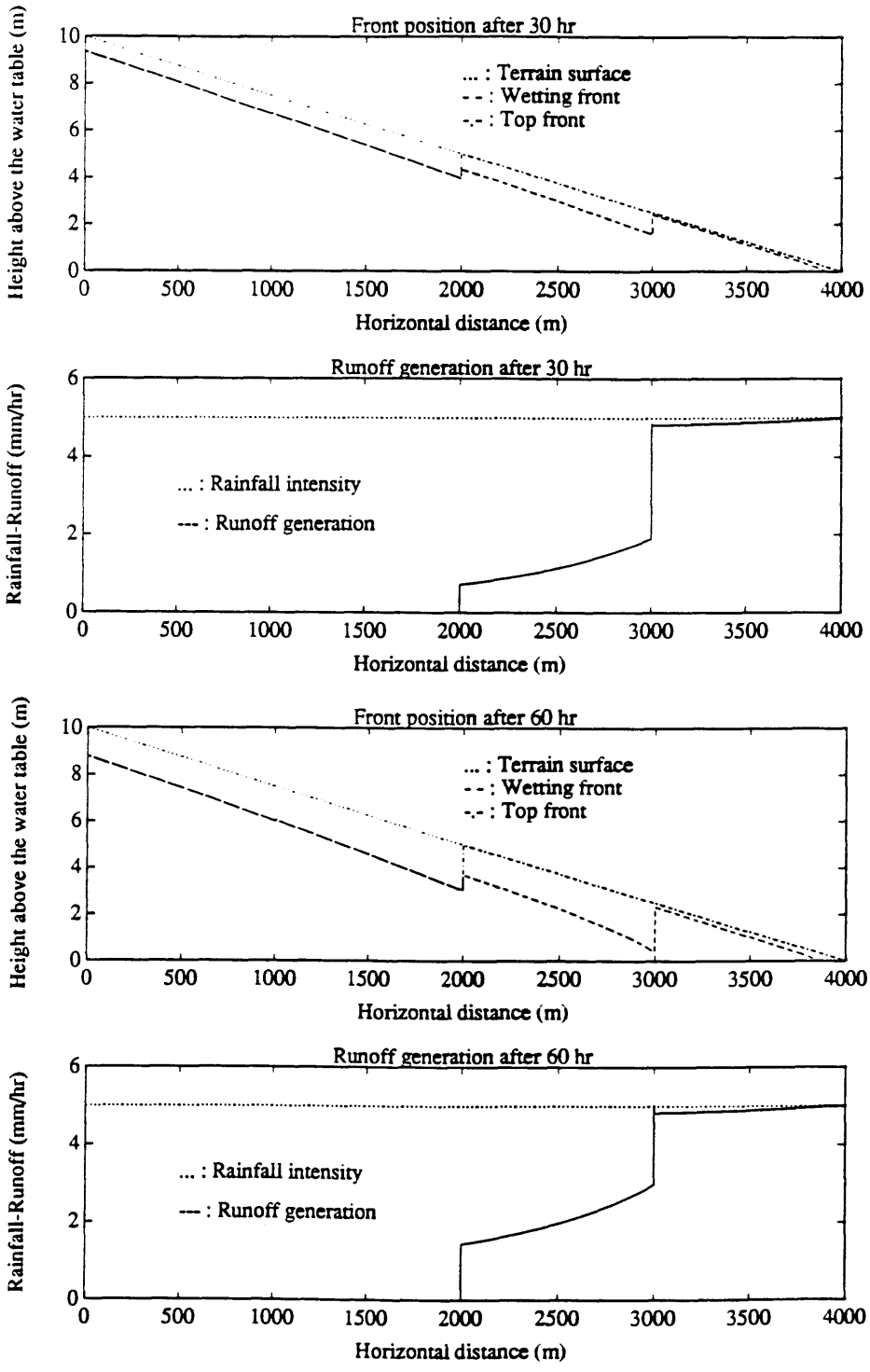


***Figure 3.8**: Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 1.*

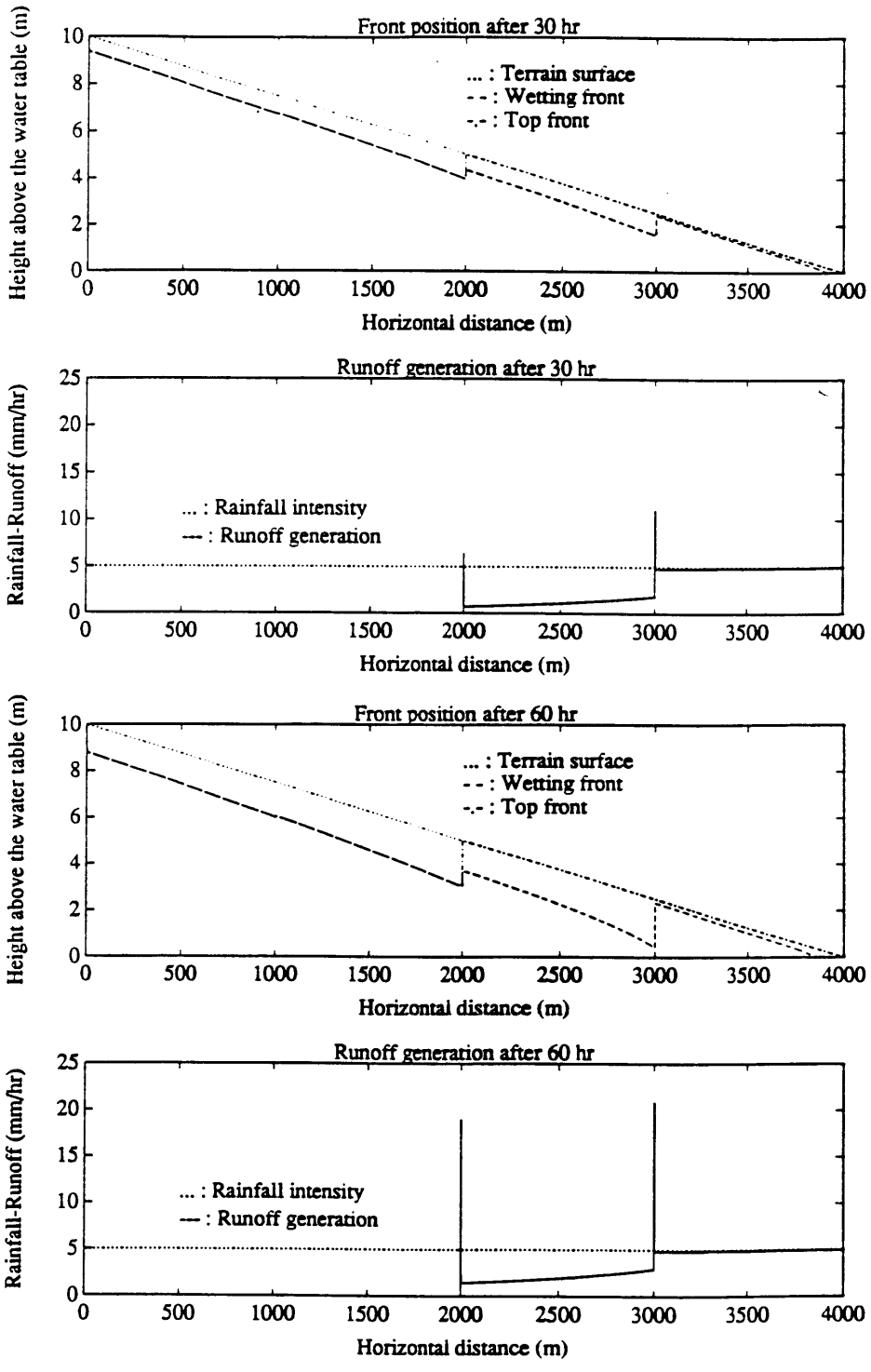


***Figure 3.9**: Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 10.*

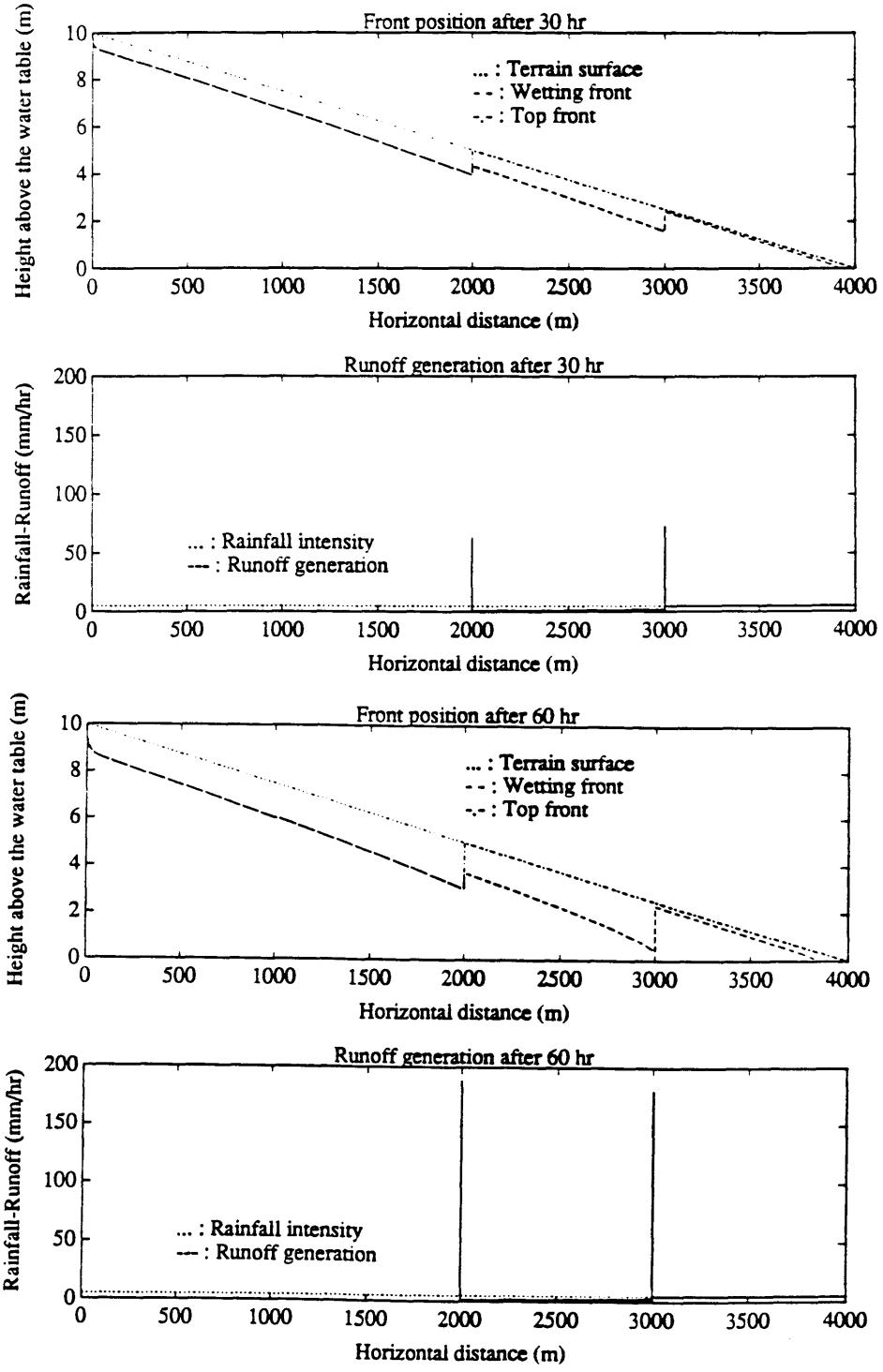


***Figure 3.10**: Front position and runoff generation for the hillslope model b after 30 and 60 hours of simulation. The anisotropy ratio is equal to 100.*

two upper soil zones are unsaturated. Wetting and top front coincide and, since there is no saturation at the surface, runoff generation is controlled exclusively by surface saturated hydraulic conductivity. Pixels in the two lower zones are saturated and infiltration capacity for them is an average of the infiltration capacity in the saturated depth. This can be clearly seen in the third soil area, where the surface saturated hydraulic conductivity (5.1 $mmhr^{-1}$) is slightly greater than the rainfall rate (5.0 $mmhr^{-1}$). As the saturated area grows in depth the effective infiltration capacity is reduced, and the area generates significant runoff volumes. In the fourth area there are also a few pixels in the lower part of the hillslope where the wetting front has reached the water table. Those pixels are fully saturated, and their infiltration capacity is null. A fourth type of runoff generation mechanism, return flow, can also be observed in the figures. Wherever there is a transition from an area with a certain soil type to another area which is saturated at the surface and has a soil type of lower hydraulic conductivity, there is a positive unbalance between inflows to and outflows from the pixel, and the difference becomes return flow. There is a local generation of runoff in the transitions from higher to lower hydraulic conductivity. That return flow appears as a spike in the runoff-generation plots, and roughly amounts to the volume of subsurface lateral flow in the hillslope, which is interrupted by the saturation state of the downslope area. The local runoff generation is higher for higher anisotropy ratios, because higher anisotropy ratios originate higher lateral flows. The effect is dramatic in the case of a_r = 100 (Figure 3.10), where runoff rates of return flow at the boundary pixels are almost two orders of magnitude larger than infiltration-excess runoff rates.

3.1.2 Basin model

The goal of the sensitivity analyses at the basin scale is to identify the importance of the different model parameters in the global hydrograph of the basin. The sensitivity analysis also gives an insight on the importance of the different runoff mechanisms at the basin scale and on the role of topography on the generation of the basin response. The parameters analyzed are:

- Initial recharge rate (to describe initial condition) R_i.
- Rate of change of hydraulic conductivity with depth f.
- Anisotropy ratio a_r.
- Routing parameters K_v, c_v and r.

The variables analyzed were:

- Total output hydrograph
- Spatial pattern of runoff generation
- Time evolution of the proportion of runoff from different sources

The Sieve basin

The catchment selected for the analysis is drained by the Sieve river, one of the tributaries of the Arno river. The Arno river crosses the cities of Florence and Pisa and drains an area of 8000 Km^2 on the northwest of the Italian peninsula. The confluence of the Sieve and the Arno is located near Florence, and the area of the Sieve catchment is approximately 840 Km^2. The basin is highly mountainous, with an average elevation of 470 m above sea level. The elevation of the highest peak is 1657 m, and the outlet is at 50 m above sea level. The basin is elongated in shape, with the

main river following the Southeast direction. Mountain ranges surround the basin to the North and the East. Figure 3.11 represents the boundaries and topography of the basin.

The rainy season lasts from October until April, with peaks in the months of November and February. Winter storms are usually originated by frontal systems. During the fall, the mountains may also produce orographic precipitation events when moist air is advected by westerly winds from the nearby Mediterranean sea. Meteorological and physical data available for the Sieve basin are described in Section 3.2.1, where the

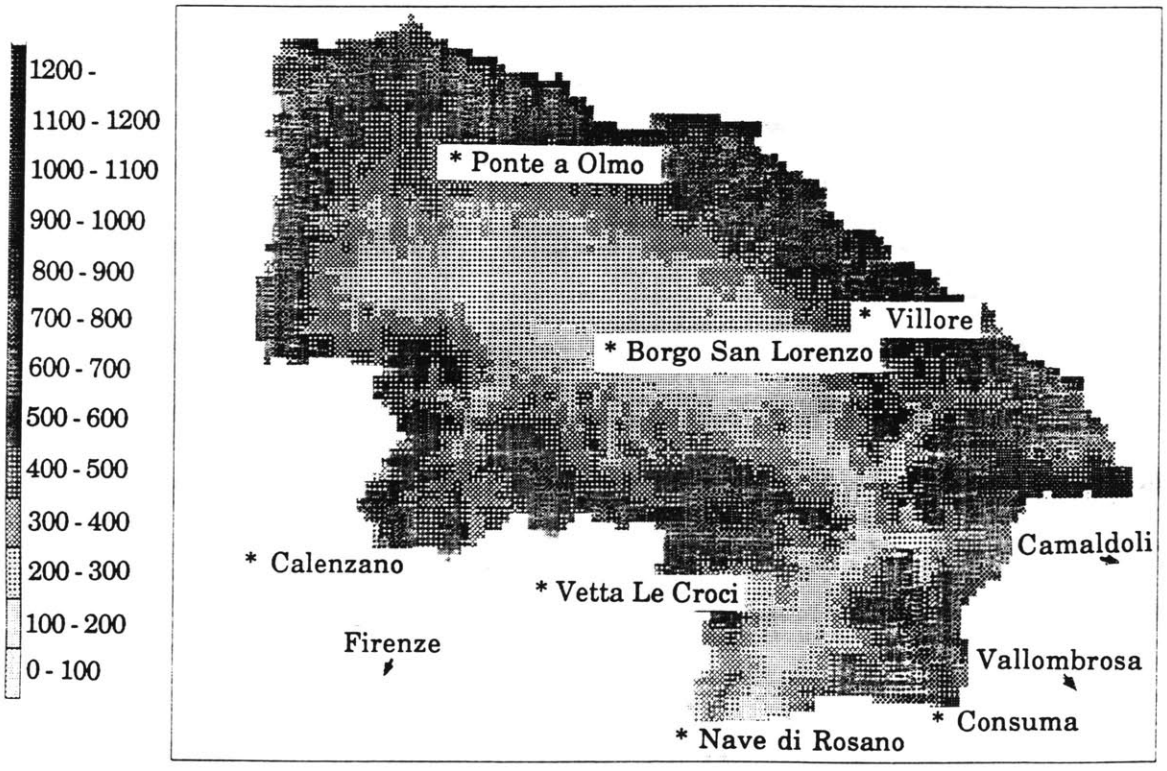


***Figure 3.11**: Boundaries and topography of the Sieve river basin.*

model is calibrated for that basin. Here we focus only on sensitivity analysis to model parameters, and, except for the topographical description of the basin, the values assigned are arbitrary.

A set of computer runs was made to test sensitivity to the list of parameters mentioned in the introduction. The different runs are all the combinations of the options summarized in Table 3.3. Four different values of the parameter f and the anisotropy ratio a_r were considered. The sixteen combinations were run for different initial states. The initial state is defined by the position of the water table $N_{wt}(x,y)$ and the definition of the initial recharge rate R_i.

***Table 3.3** Parameter values*

Parameter				
Parameter f (mm h^{-1})	1×10^{-4}	3×10^{-4}	5×10^{-4}	7×10^{-4}
Anisotropy ratio a_r	1	10	100	500
Initial water table (prob. exceed.)	10%	50%	90%	
Parameter R_i (Interpretation as)	Moisture content	Recharge rate		

Initial water table positions were generated using the model described by Cabral et al., (1990), which considers only water movement in the saturated zone below the water table. The model gives the water table position for which the baseflow drained from the basin is in equilibrium with a long term recharge rate, assumed constant throughout the basin. Three different initial states were generated for each combination of parameters (anisotropy ratio a_r and parameter f). The initial states correspond to equilibrium situations for interstorm baseflows that are exceeded 10%, 50% and 90% of the years respectively for

the month considered (November). The 10% option corresponds to a wet state, the 50% option to an average state, and the 90% option to a dry state. The initial states were generated for the four different values of the parameter f, but only two values of the anisotropy ratio were considered, because values greater than 10 lead to unrealistic water table profiles. It was considered that the anisotropy ratio used in DBS refers only to the top layers of the soil, where the effects of vegetation and weathering create the strongly layered structure that causes high anisotropy ratio. The water table model deals with the deep layers of soil below the water table, where strong anisotropy is more unlikely to occur. Therefore, it was decided to generate initial states only with anisotropiy ratios of 1 and 10 and apply the initial state generated with a_r=10 to the cases a_r=100 and a_r=500.

The infiltration model is based on the kinematic approximation, which assumes a direct relationship between moisture content and vertical flow. In reality, moisture gradients in the vertical create capillary forces which balance the gravitational gradient, and an inter-storm equilibrium moisture profile develops, where water moves very slowly driven by surface evaporation. This situation cannot be captured by the kinematic model, and therefore, inter-storm moisture descriptions under the kinematic approximation are always imperfect. If the kinematic parameterization is chosen to reproduce the real moisture content, the model overestimates flows, because it does not account for capillary forces. Conversely, if the model is chosen to reproduce flows, moisture contents are underestimated. An interesting dilemma arises as to which interpretation to choose for the description of the initial moisture content in the basin scale model: moisture content or infiltration rate. An important part of the sensitivity analysis was dedicated to study this

aspect. Two options were considered to define the initial recharge rate, each corresponding to a different interpretation of the meaning of the parameter R_i.

In the first case, R_i is interpreted as a description of the moisture content of the soil column at the beginning of the simulation. The initial infiltration rate is defined in such a way that the moisture profile reaches saturation at the water table level. Since the water table is located at different depths for different grid elements, R_i is not homogeneous throughout the basin. R_i is higher in those elements were the water table is shallower, and smaller in elements where the water table is deep. Infiltration values obtained in this case should not be interpreted at their face value, but as a surrogate of the variable that they represent, namely the moisture content.

In the second option, R_i is interpreted as the real recharge rate during the previous inter storm period. In this case, R_i is obtained from the inter-storm baseflow. The baseflow is in equilibrium with a recharge rate given by $R_i = \frac{3.6 q_{bf}}{A}$, where R_i is the initial recharge rate in mmh^{-1}, q_{bf} is the baseflow in m^3s^{-1} and A is the watershed area in Km^2. The value of R_i is assumed to be uniformly distributed throughout the basin. In this interpretation only the value of infiltration rates has real meaning.

Sensitivity to parameter f

Figures 3.12 to 3.14 represent the sensitivity of the total hydrograph to the parameter f. Each figure contains four plots corresponding to the different anisotropy ratios considered. Each plot compares the total

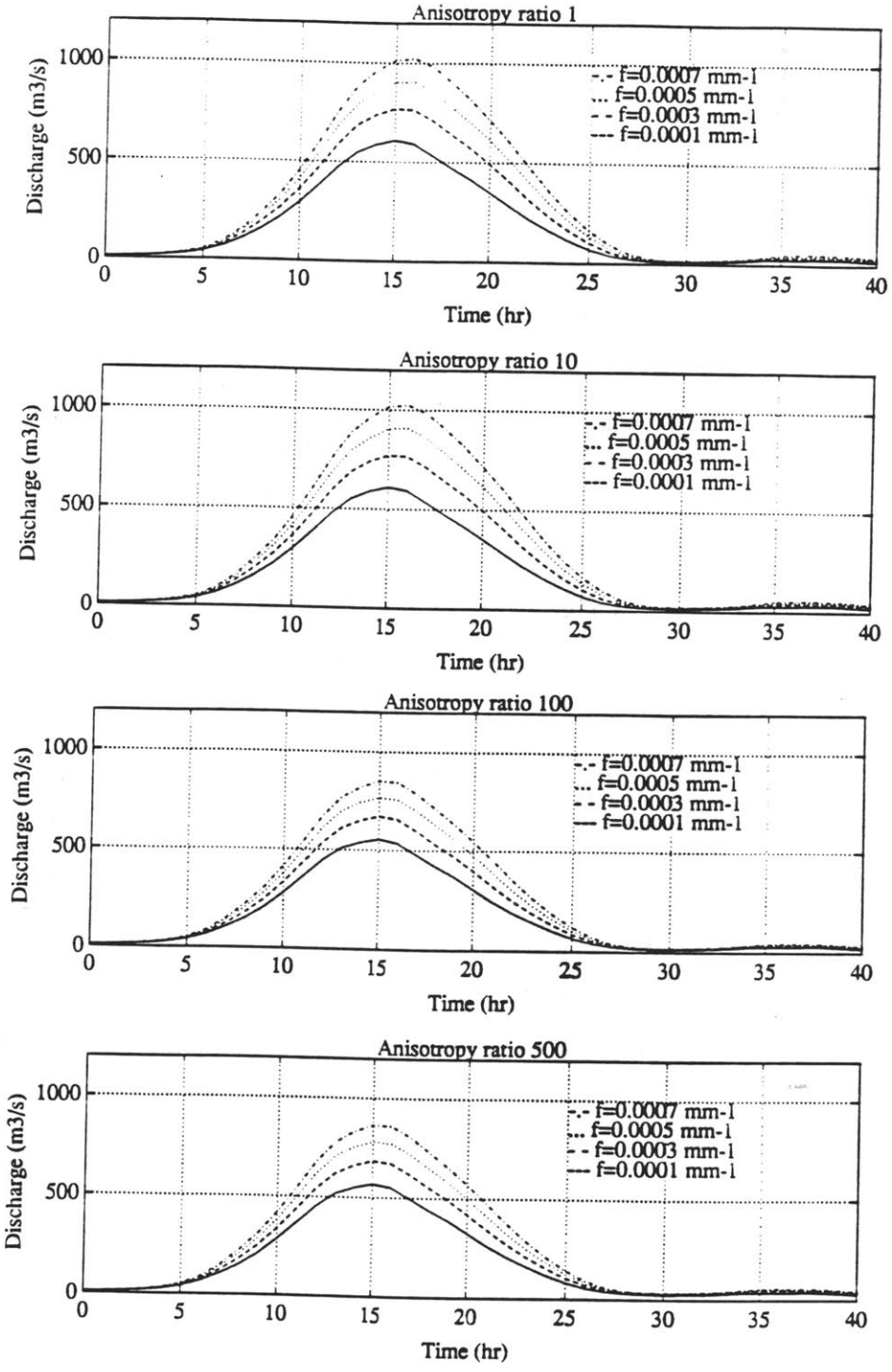


Figure 3.12a: *Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 10% probability of exceedance and the parameter R_i interpreted as initial moisture content.*

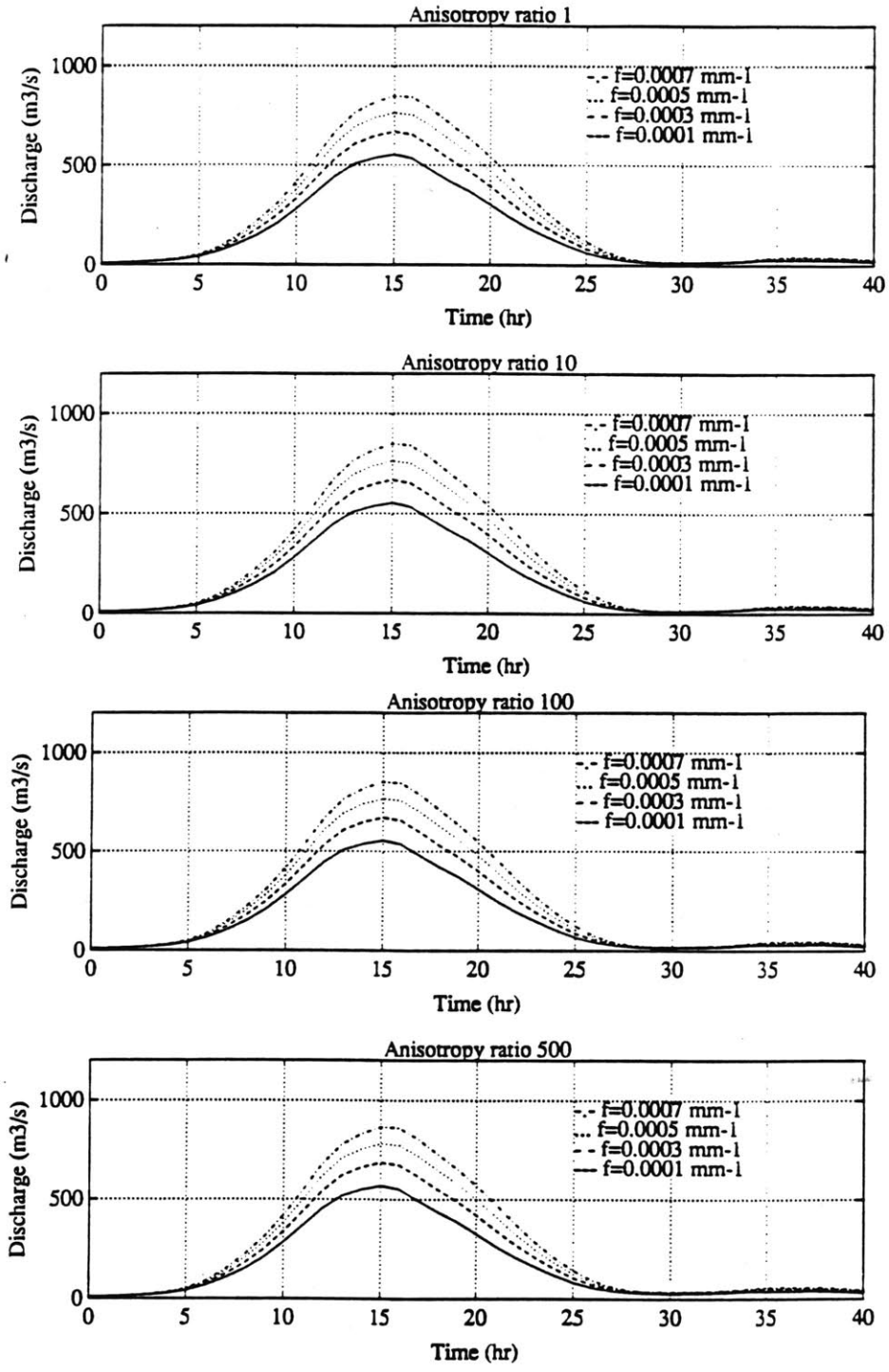


Figure 3.12b: *Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 10% probability of exceedance and the parameter R_i interpreted as initial recharge rate.*

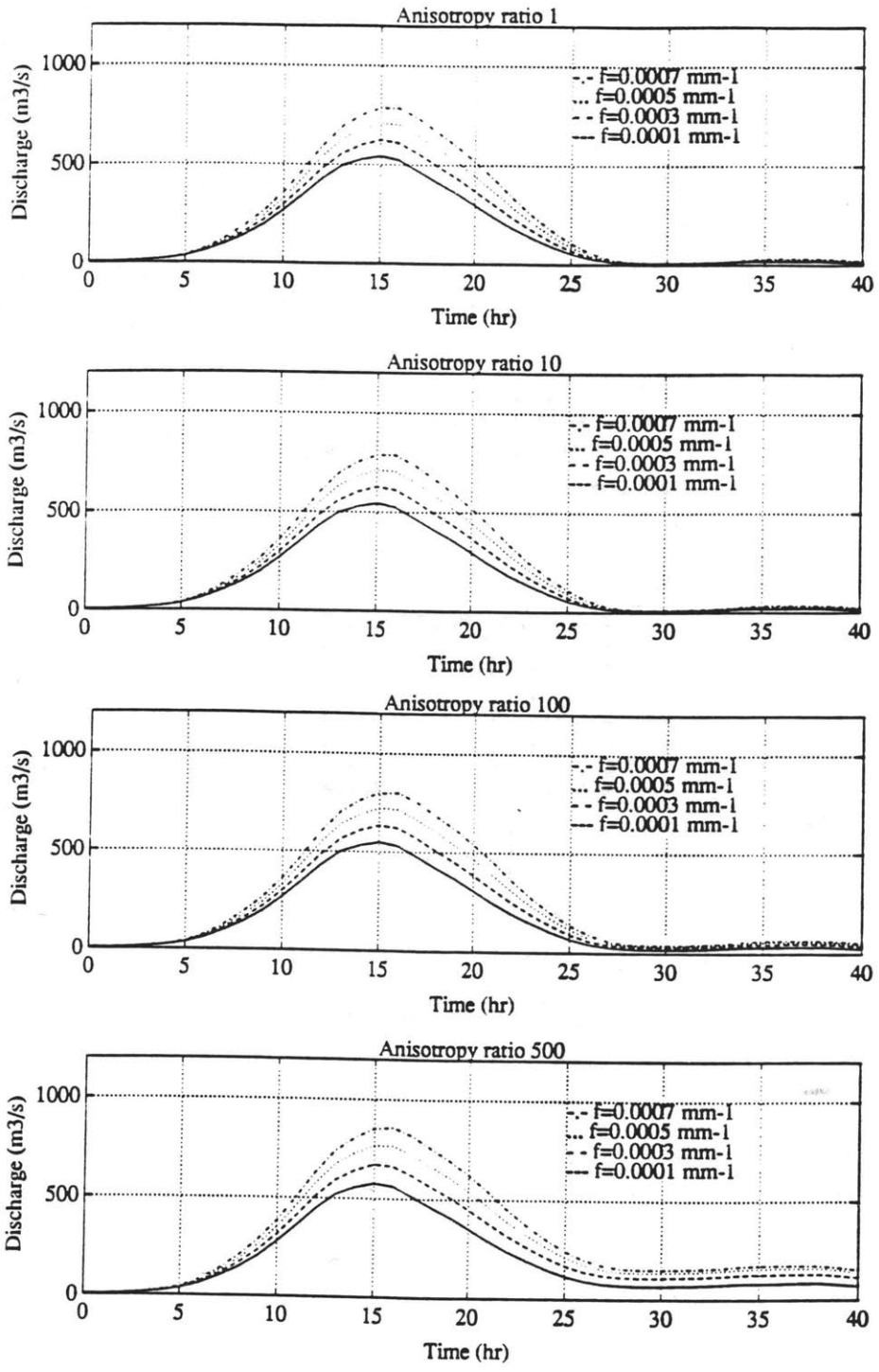


Figure 3.13a: *Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 50% probability of exceedance and the parameter R_i interpreted as initial moisture content.*

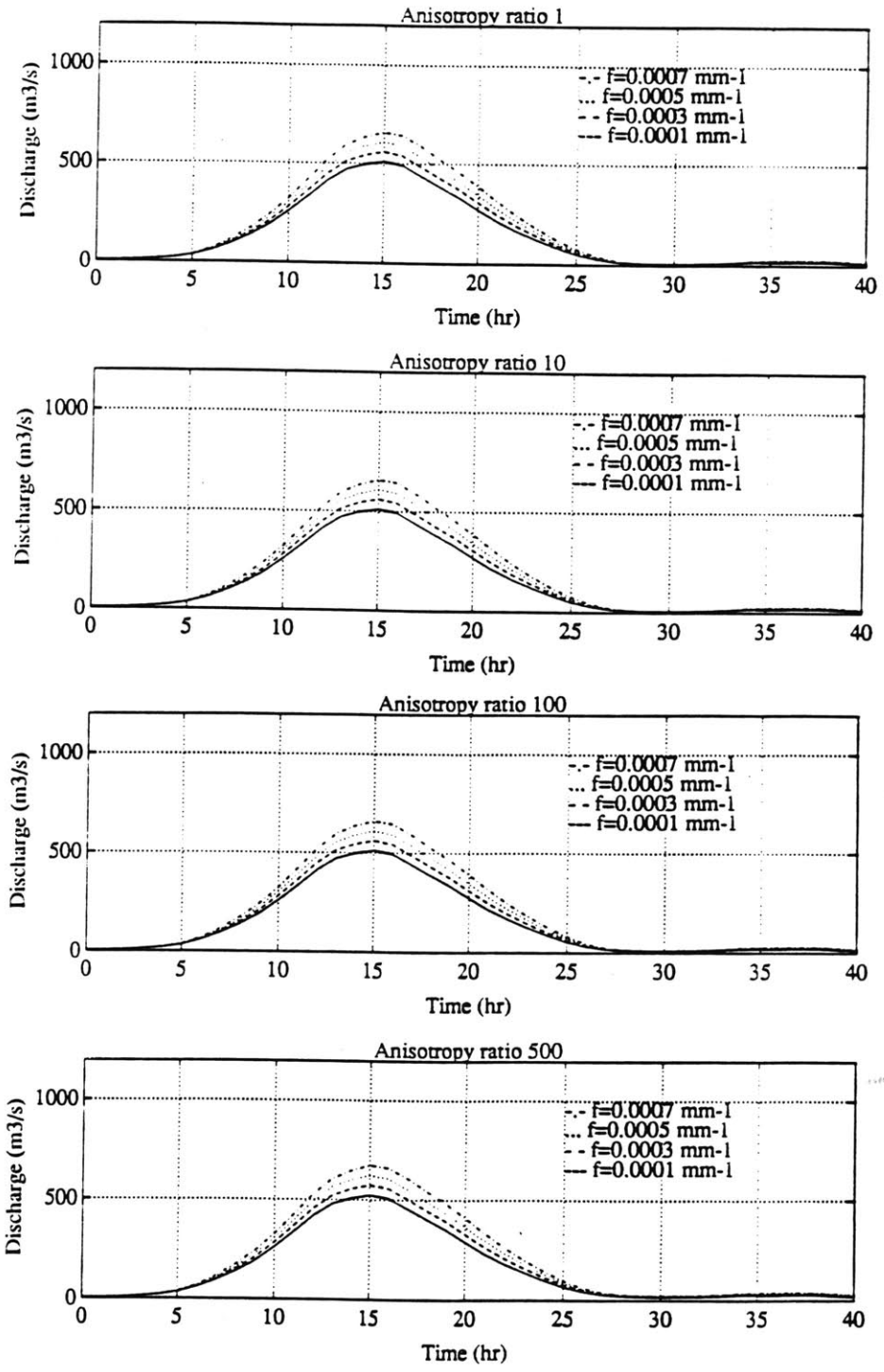


***Figure 3.13b**: Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 50% probability of exceedance and the parameter R_i interpreted as initial recharge rate.*

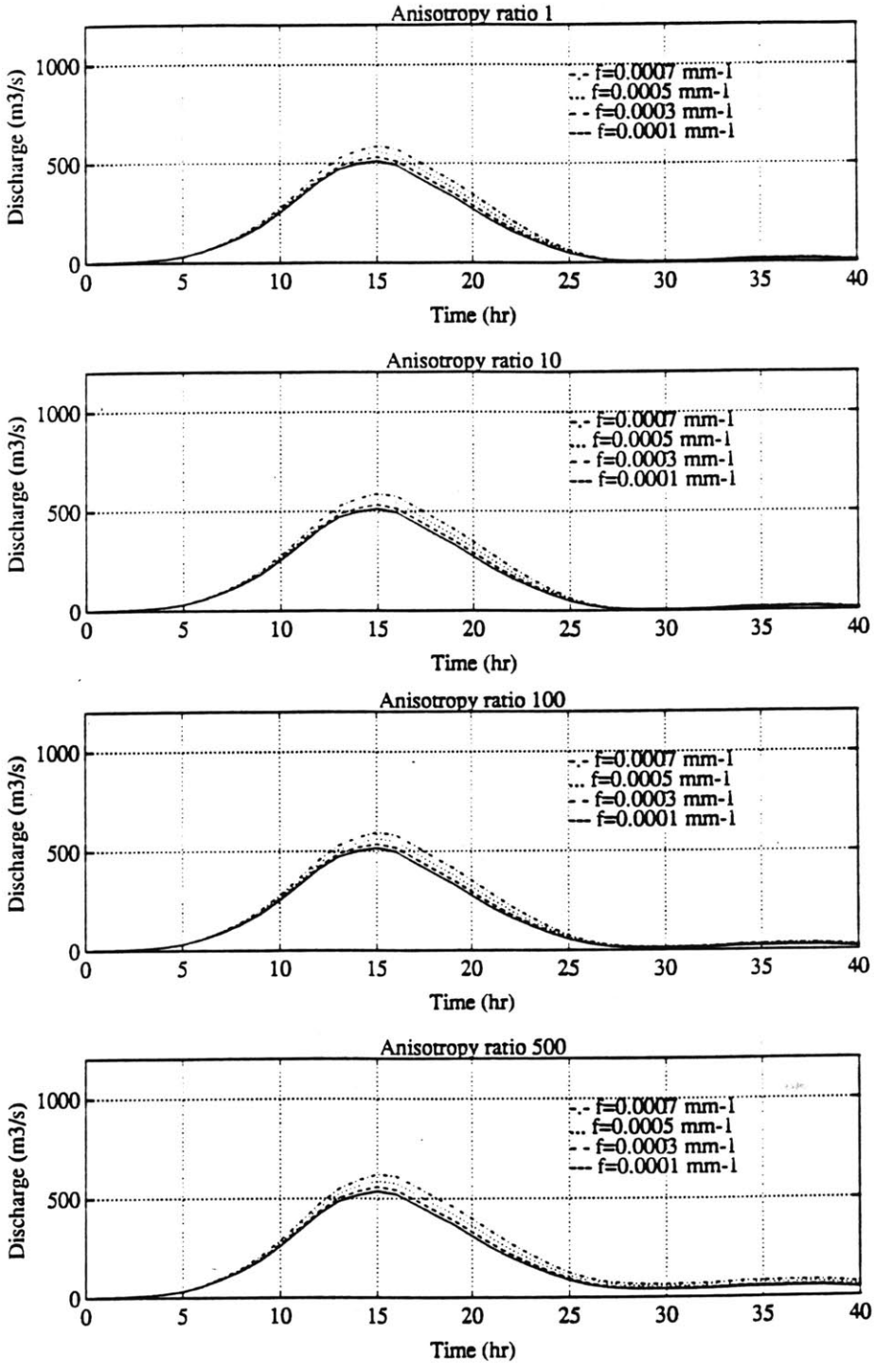


Figure 3.14a: *Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 90% probability of exceedance and the parameter R_i interpreted as initial moisture content.*

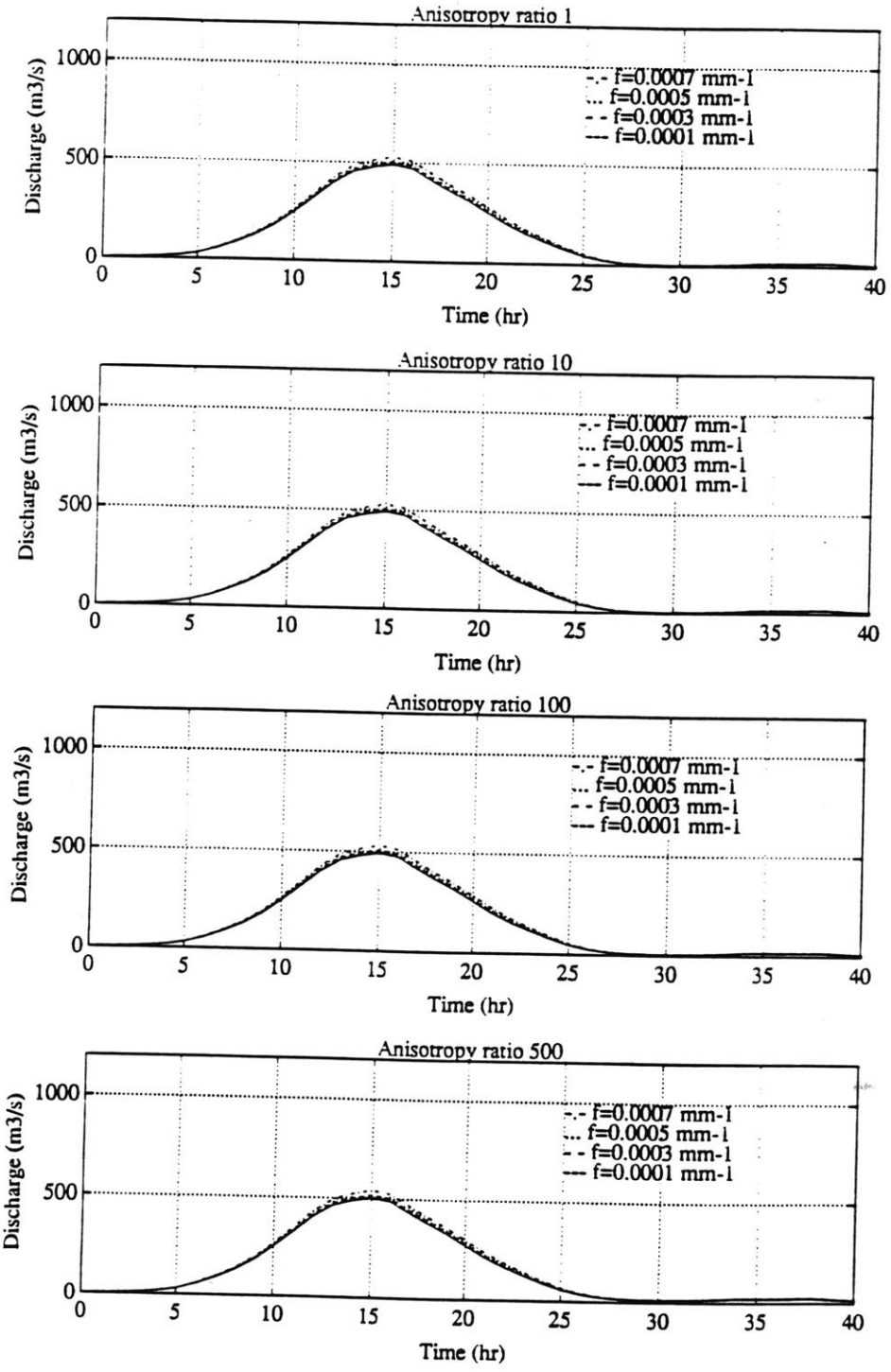


***Figure 3.14b**: Sensitivity of basin response to parameter f for anisotropy ratios of 1, 10, 100 and 500. The results correspond to the initial state with 90% probability of exceedance and the parameter R_i interpreted as initial recharge rate.*

hydrograph for the four values of f. Figures 3.12a and 3.12b correspond to the wet initial state, figures 3.13a and 3.13b to the average initial state and figures 3.14a and 3.14b to the dry initial state. Figures labelled 'a' correspond to the initial state option where R_i is interpreted as the initial moisture content, and figures labelled 'b' correspond to the initial state option where R_i is interpreted as the initial recharge rate. In general, model sensitivity to the parameter f is very strong. The sensitivity obtained is greater if the parameter R_i is interpreted as the initial moisture content, since it leads to higher initial moisture in the basin. The influence of the initial position of the water table is also remarkable. The sensitivity is higher in the wet and average cases, and significantly lower in the dry case. The effect of the anisotropy ratio on the sensitivity to parameter f is less clear, and it can only be noticed in the wet initial state case.

Figures 3.15a and 3.15b represent the time evolution of runoff generation in the basin compared to the rainfall rate. Figures 3.16a and 3.16b represent the effect of the different runoff generation mechanisms on the total hydrograph obtained for the basin. There are four plots in each figure, each corresponding to a different value of f. The rest of the parameters correspond to the base case. These four figures offer a more detailed description of the cases represented in the second plot of figures 3.13a and 3.13b.

The runoff is decomposed according to its different possible origins. Two basic types of runoff are considered: infiltration excess or surface runoff and return flow or subsurface runoff. Since the dynamics of the state of the basin are also important, the plot differentiates between runoff generated in the areas of the basin which are permanently saturated

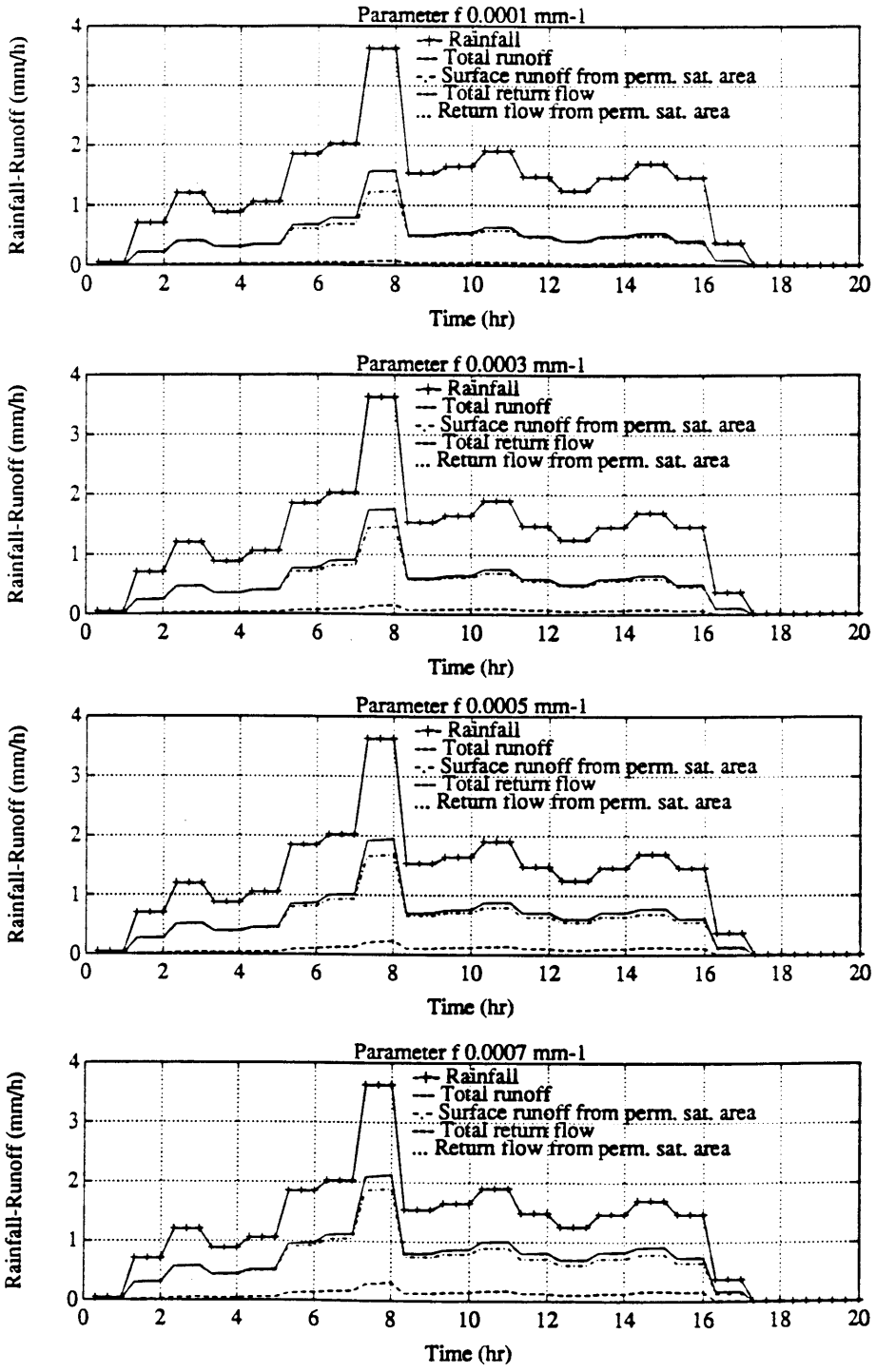


***Figure 3.15a**: Time evolution of different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content.*

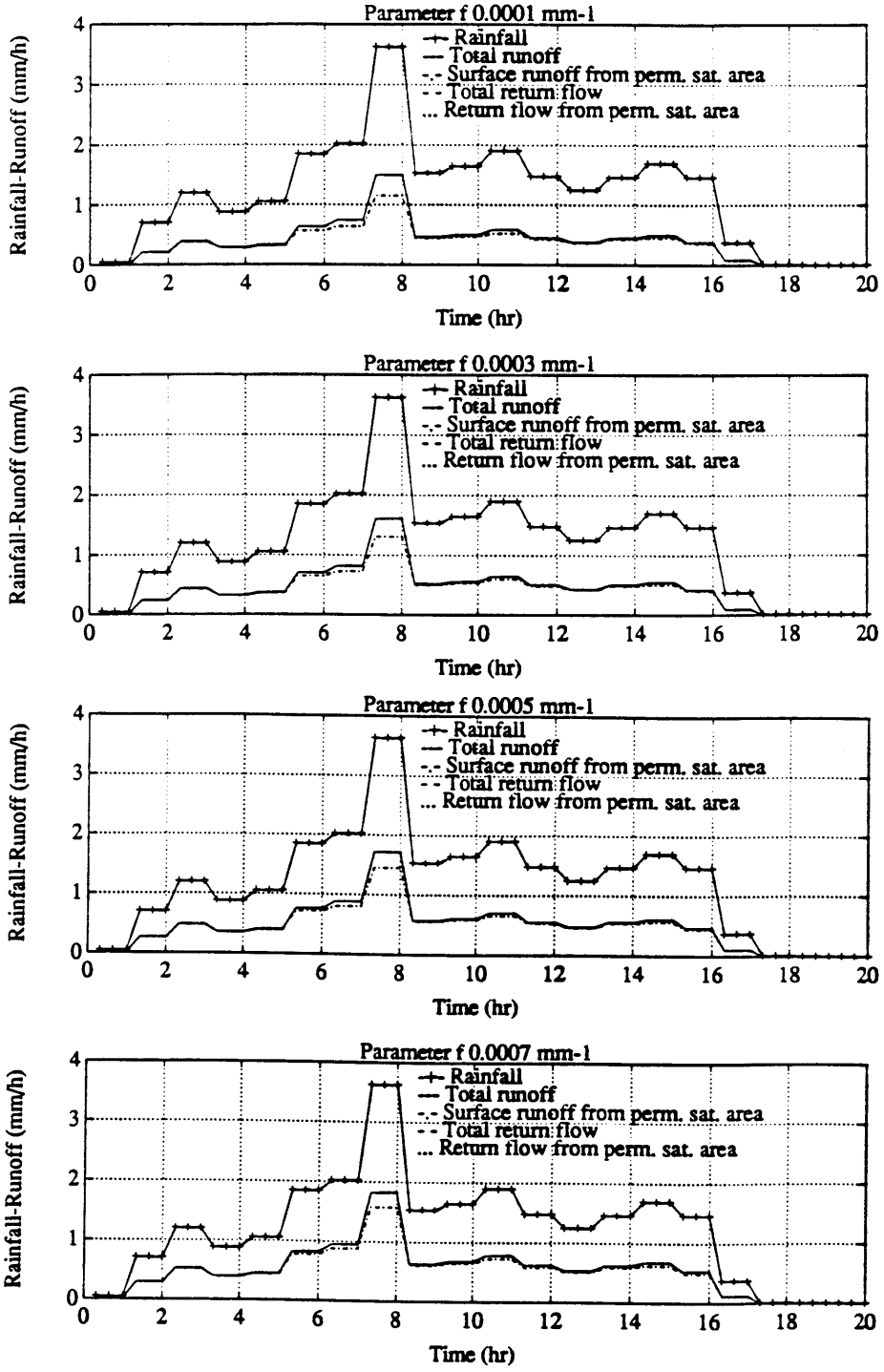


***Figure 3.15b**: Time evolution of different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

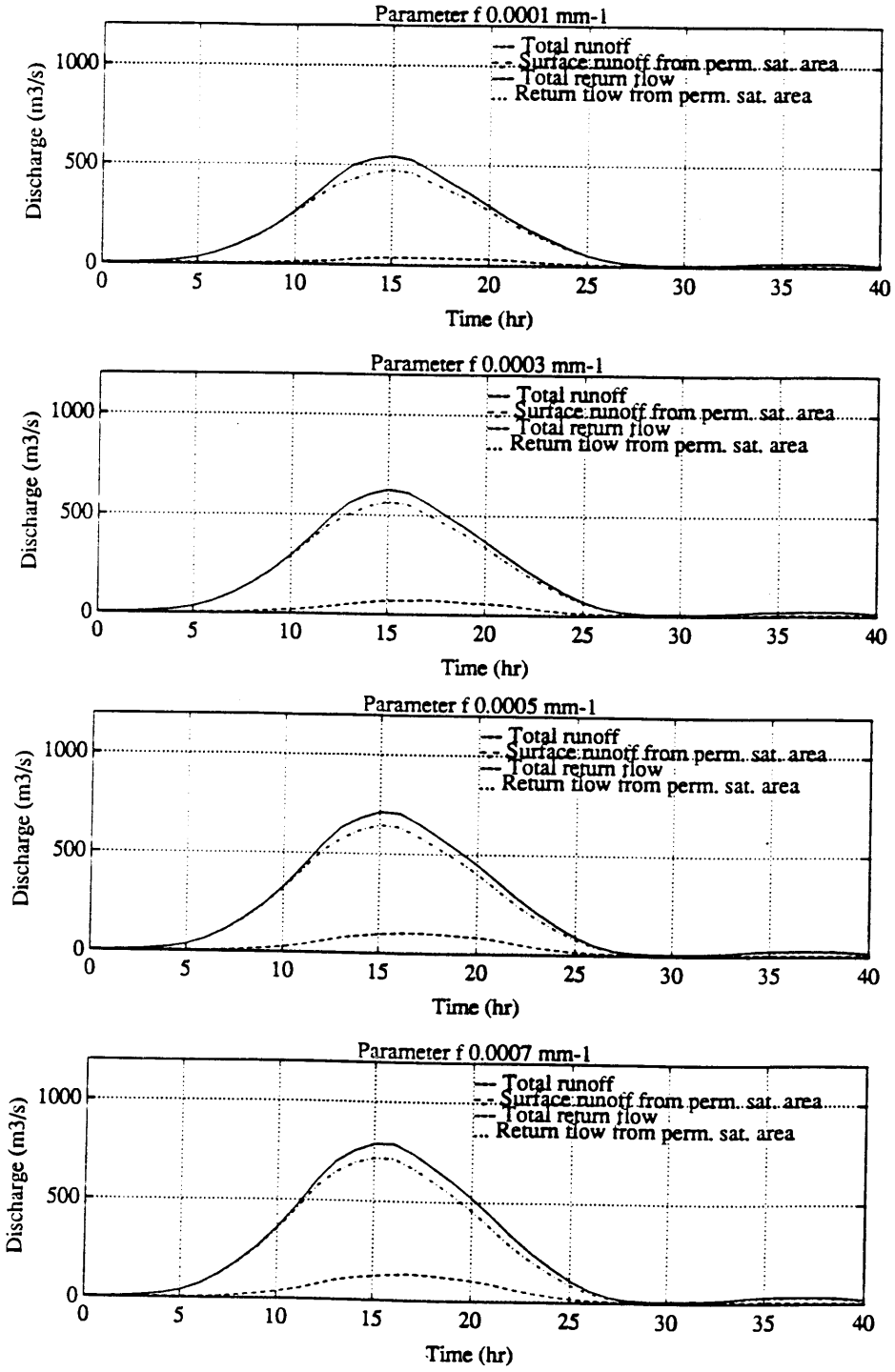


***Figure 3.16a**: Decomposition of basin response into different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content.*

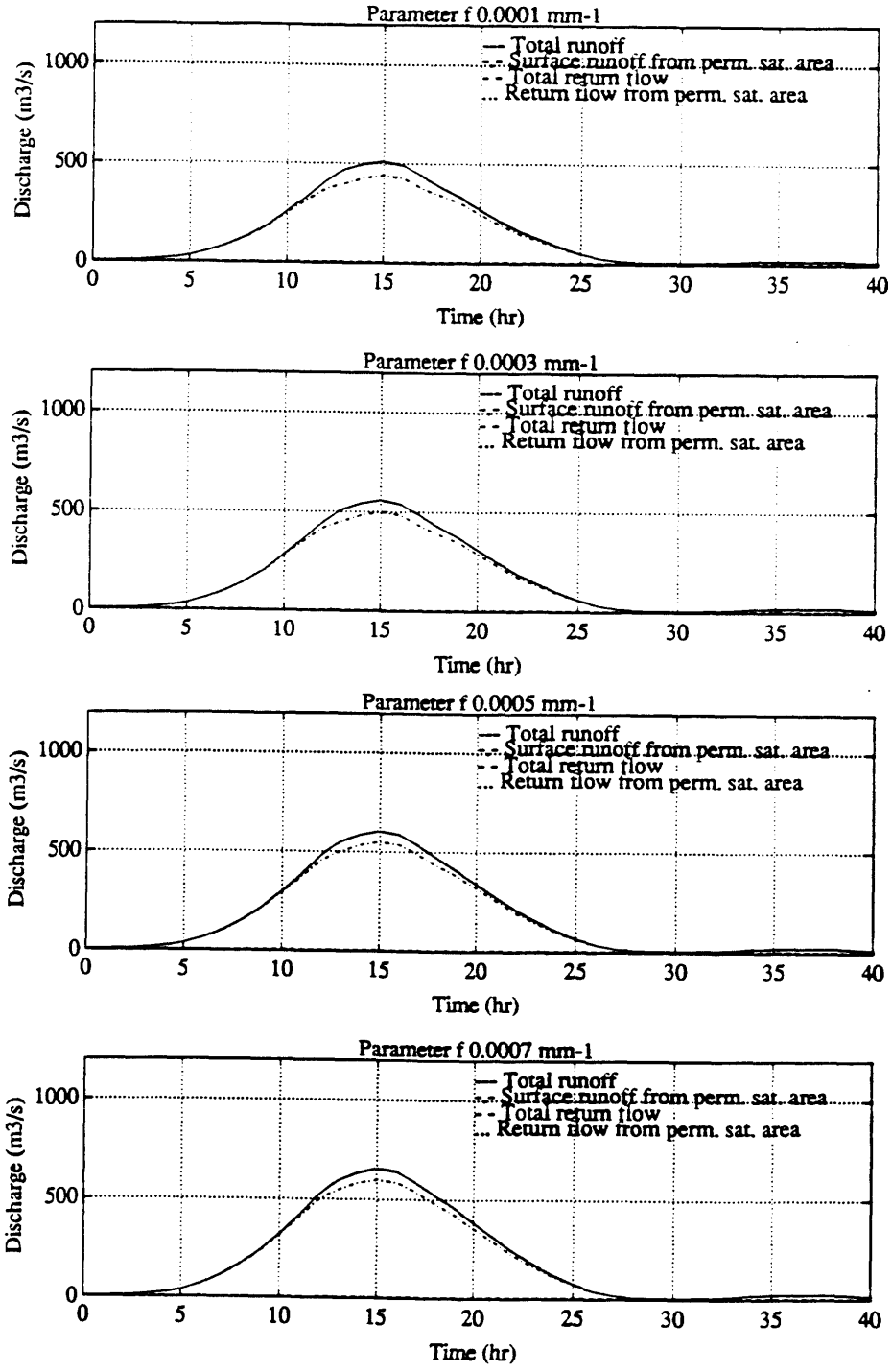


***Figure 3.16b**: Decomposition of basin response into different modes of runoff generation for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

(stream areas or areas where the initial water table is at the surface) and runoff generated in the areas of the basin which are temporarily saturated. In general, runoff generated in the areas permanently saturated is not sensitive to the parameter values, and is only affected by the initial state. Runoff generated in the rest of the basin can be extremely sensitive to parameter values, accounting for the visible differences at the basin scale.

The lowest part of the runoff plot (dotted line) represents the return flow generated in the areas permanently saturated. This area is barely noticed in almost all plots, since the contribution of this type of runoff to the global response is very small for the anisotropy ratio considered in this case. The next layer (between the dotted and the dashed line) represents the return flow generated in areas that have become temporarily saturated. The third layer (between the dashed and the dash-dotted line) represents surface runoff generated in areas permanently saturated. Since the infiltration capacity in these areas is null, these areas usually account for the most important fraction of total runoff generation. The last fraction (between the dash-dotted line and the solid line) represents the runoff generated in the rest of the basin, and includes the purely Hortonian runoff generated in the unsaturated areas and the infiltration excess runoff generated in the saturated areas.

Figures 3.15 and 3.16 show that an important part of the sensitivity of the total hydrograph to the parameter f comes from the return flow generated in the areas of the basin which become saturated during the storm. There is also a different response in the surface runoff, but it is comparatively much smaller. This second influence is isolated in figures 3.15b and 3.16b, because in the case where R_i is interpreted as initial

recharge rate the initial moisture is generally too low to generate appreciable return flow. Therefore, in this case the sensitivity to parameter f is almost exclusively due to different surface runoff.

The parameter f defines the level at which the wetting front will reach saturation for a given infiltration rate. As f becomes greater, saturation is reached at a smaller depth, and therefore a greater number of pixel elements become saturated. The infiltration capacity in those pixels is significantly reduced and they generate more surface runoff, but the most important effect is that when a certain area becomes saturated all the subsurface flow converging into it becomes return flow. The mechanisms through which the parameter f increases runoff generation are: (1) increasing the surface of locally saturated areas, and (2) enhancing the production of return flow in these locally saturated areas.

Sensitivity to anisotropy ratio

Figures 3.17 to 3.19 describe the influence of anisotropy ratio on the total hydrograph. The four plots contained in each figure correspond to the different values of f. In each plot the sensitivity to the four values of a_r considered is compared. As in the previous case, label 'a' corresponds to R_i interpreted as initial moisture content and label 'b' corresponds to R_i interpreted as initial recharge rate. Figures 3.17a and 3.17b represent the wet initial state, figures 3.18a and 3.18b the average initial state, and figures 3.19a and 3.19b the dry initial state.

Model sensitivity to the anisotropy ratio is very small in the case when R_i is interpreted as the initial recharge rate. In the case when R_i is interpreted as the initial moisture content the sensitivity is higher, but

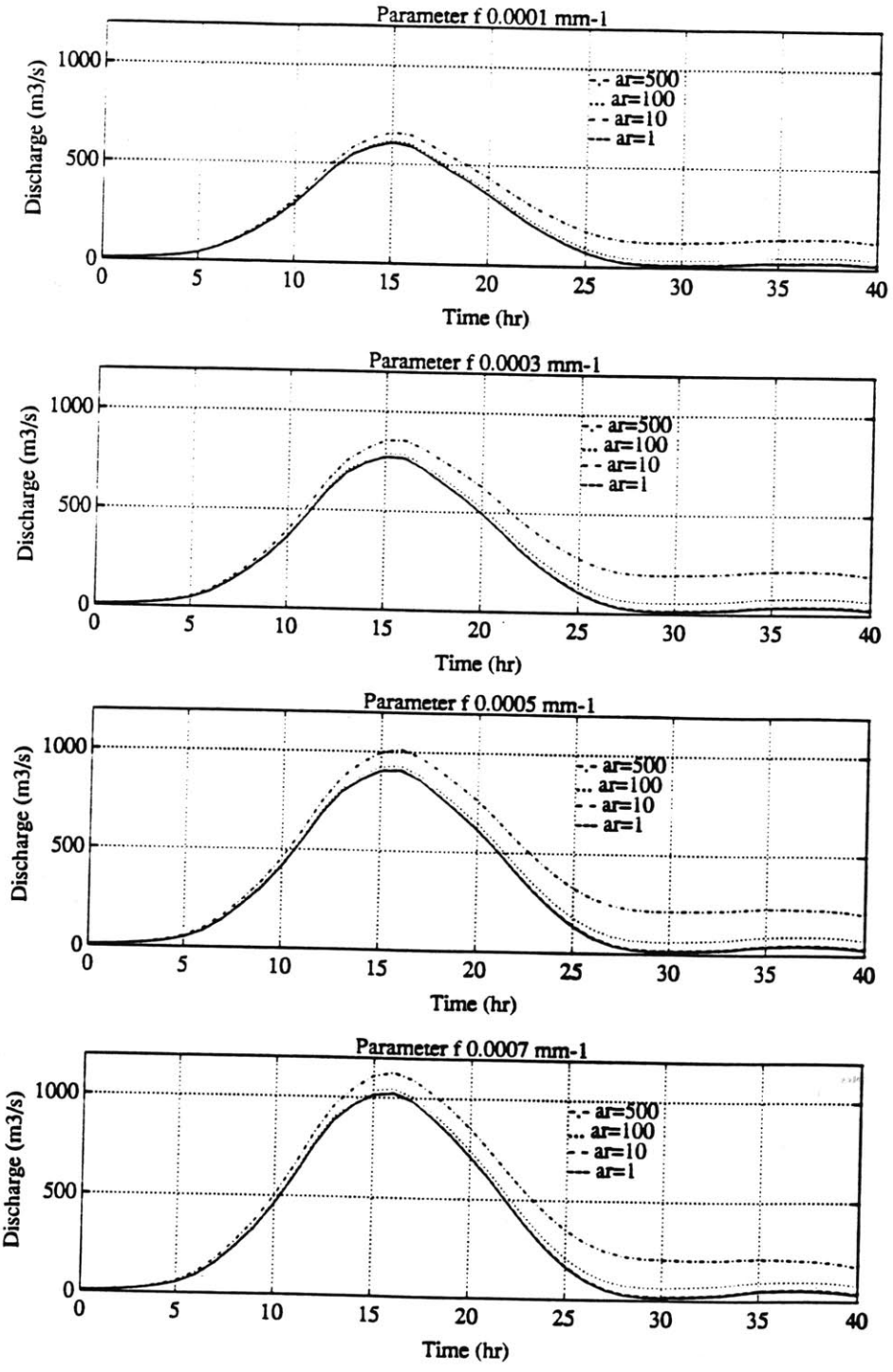


Figure 3.17a: *Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 10% probability of exceedance and parameter R_i interpreted as initial moisture content.*

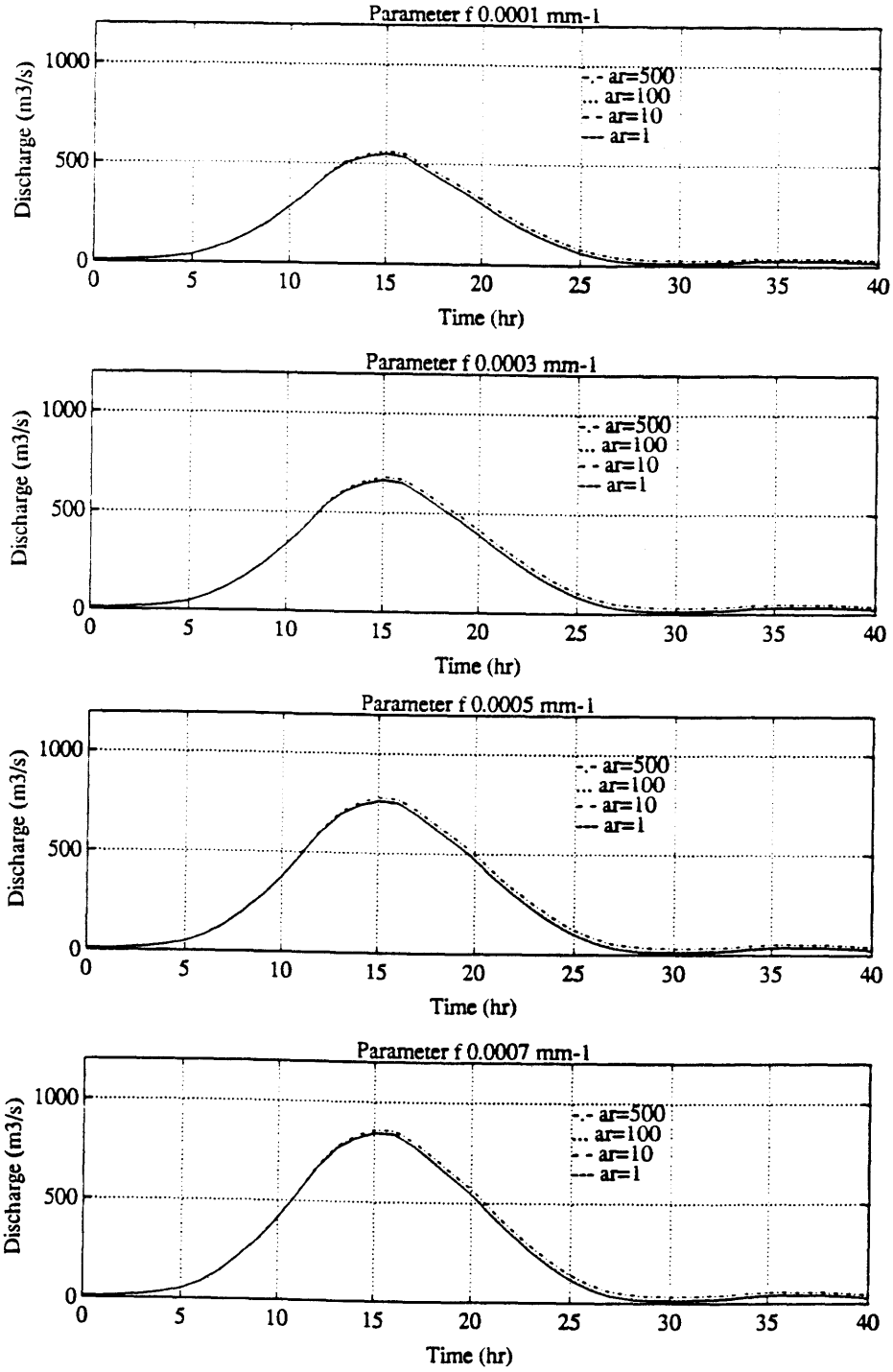


***Figure 3.17b**: Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 10% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

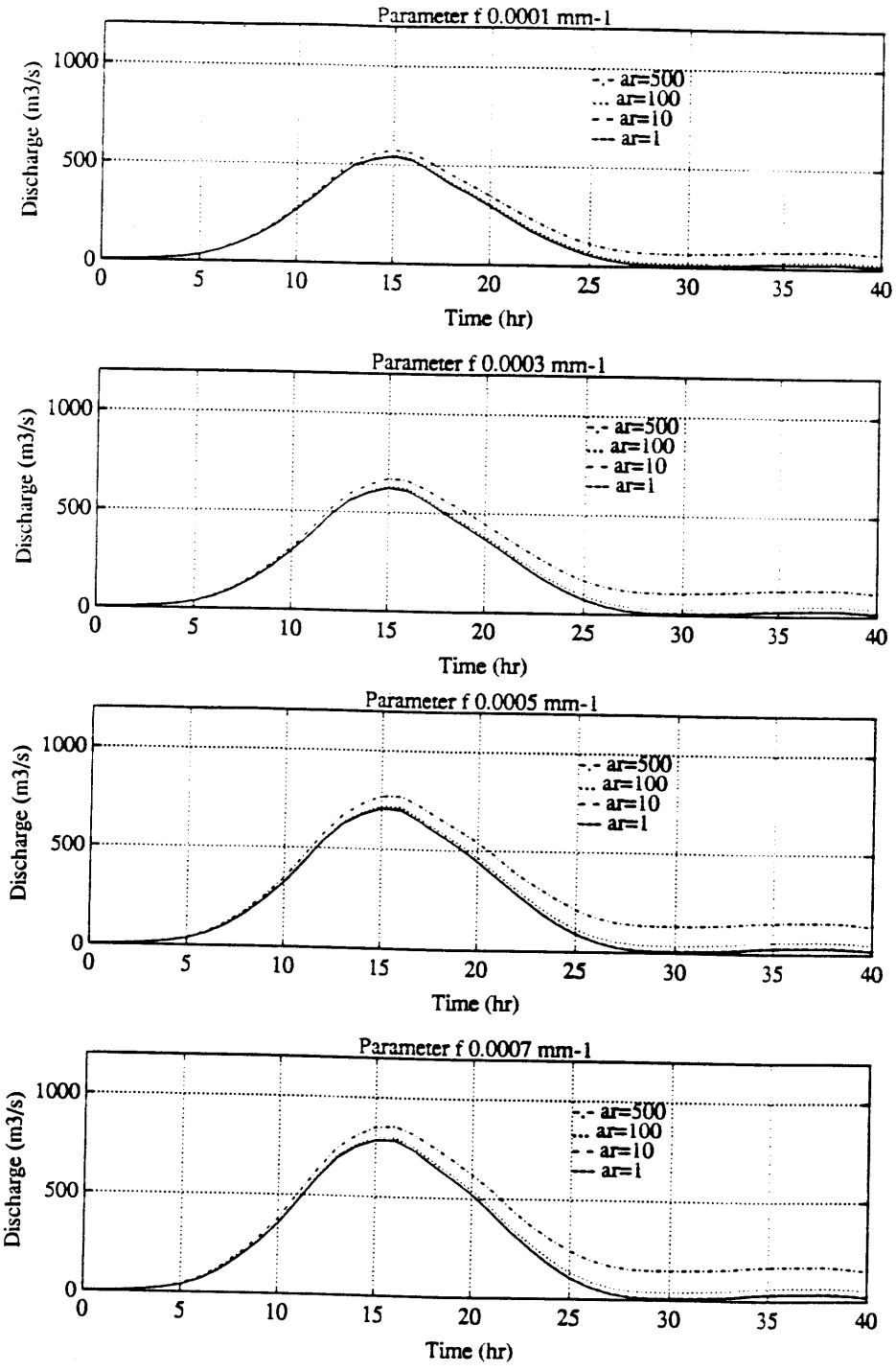


***Figure 3.18a**: Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content.*

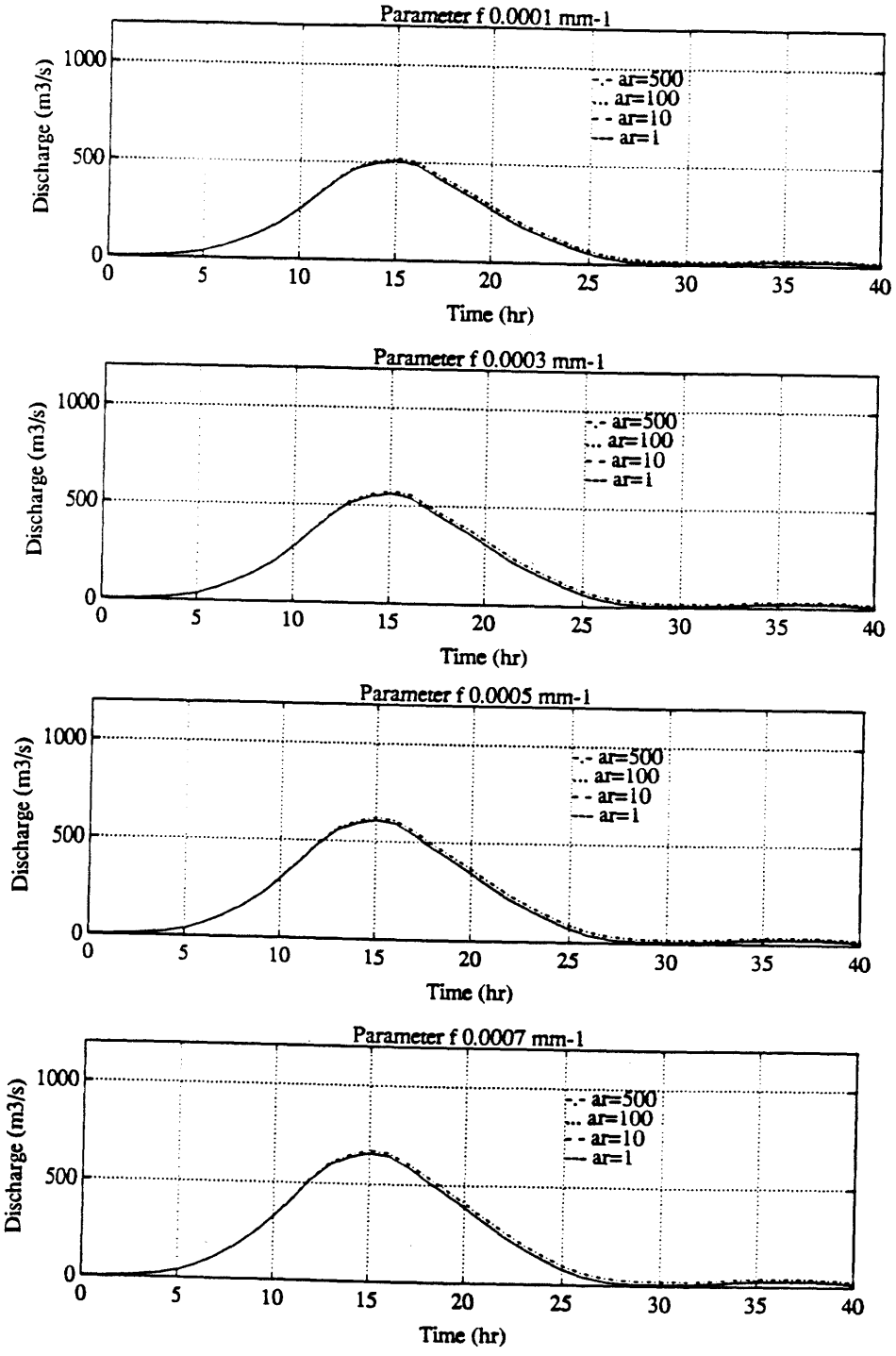


***Figure 3.18b**: Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

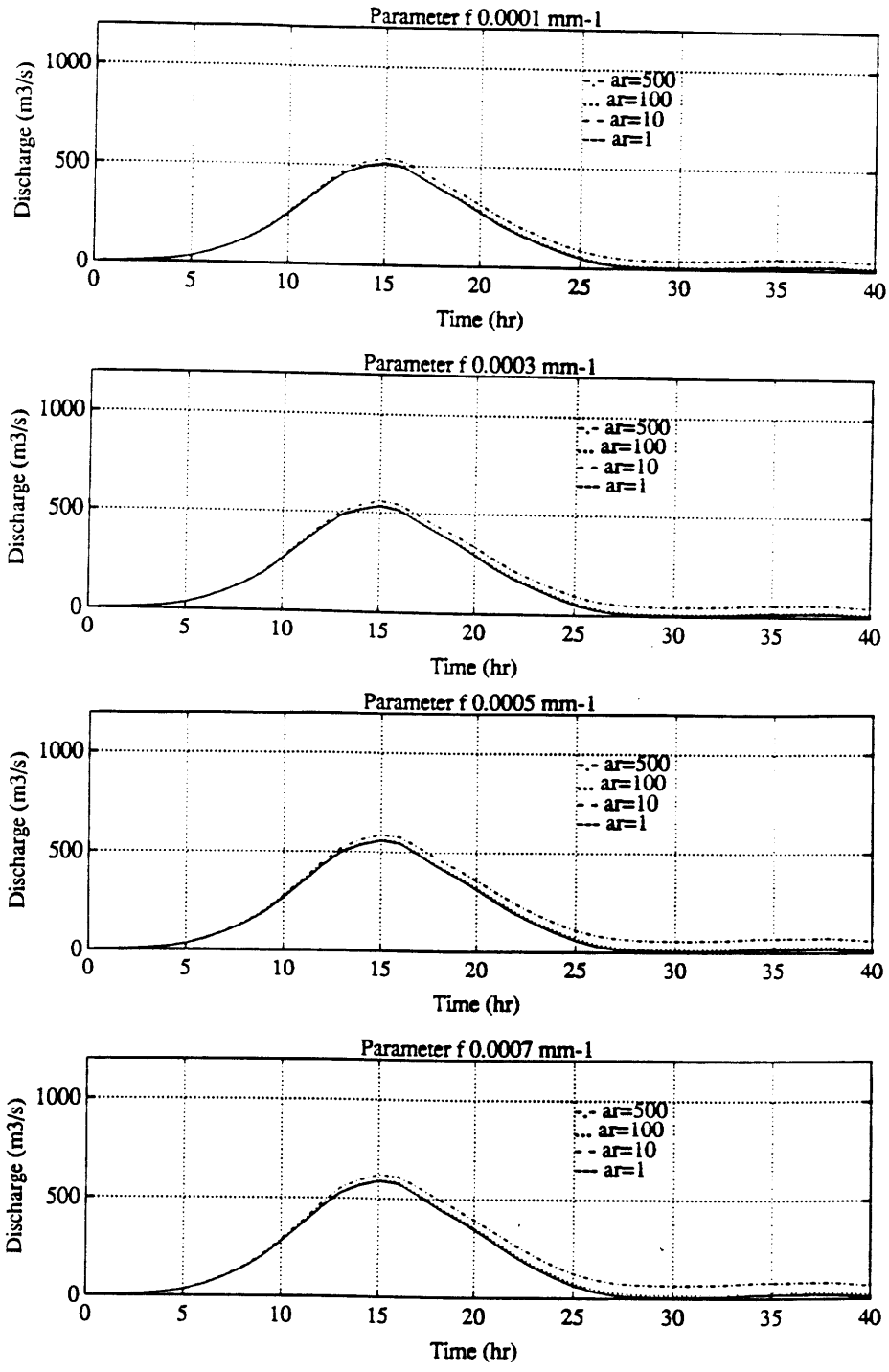


***Figure 3.19a**: Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 90% probability of exceedance and parameter R_i interpreted as initial moisture content.*

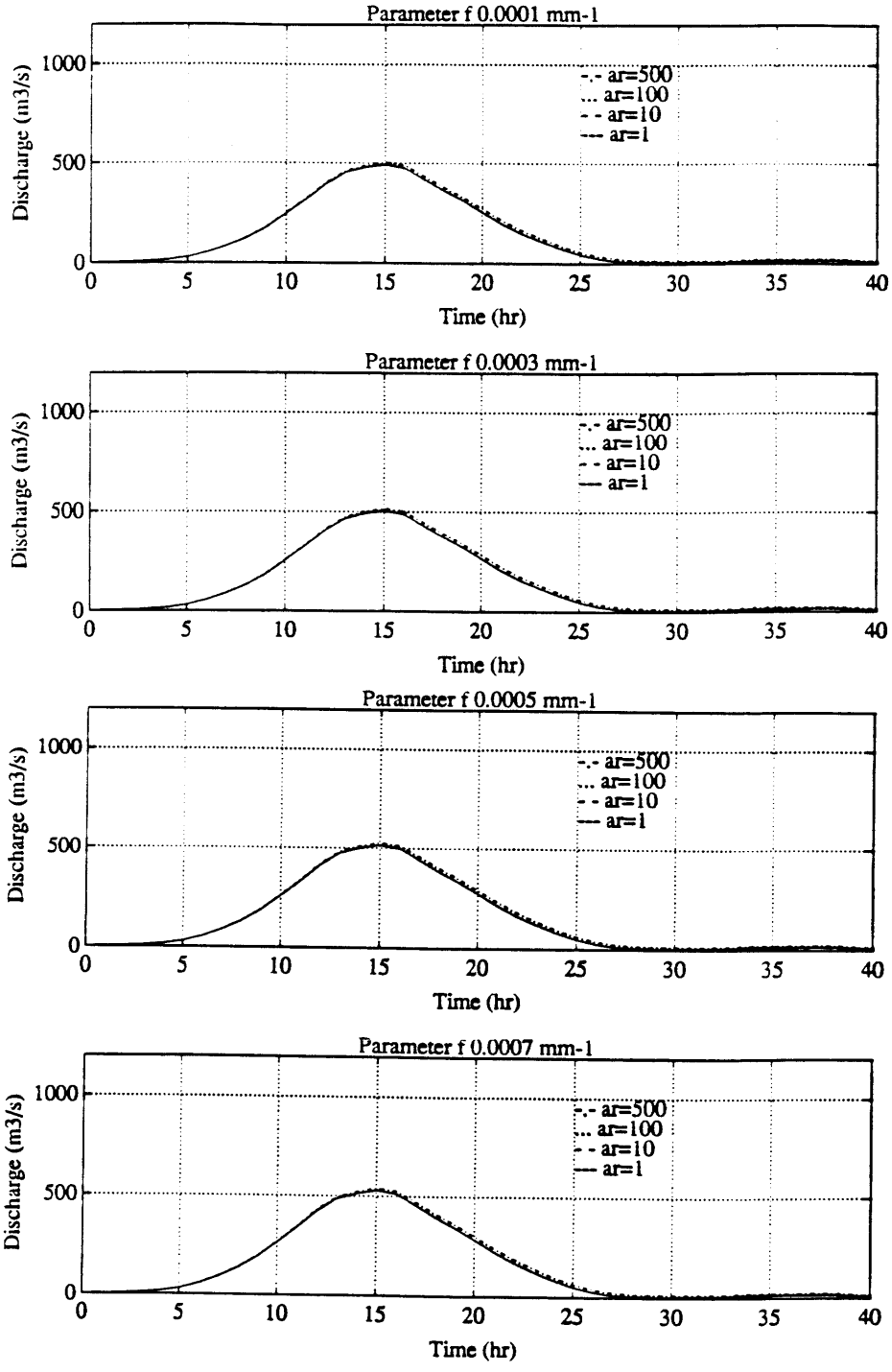


***Figure 3.19b**: Sensitivity of basin response to anisotropy ratio for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to initial state with 90% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

smaller than the sensitivity to the parameter f. In general, only the case a_r=500 appears to be significantly different from the rest. a_r=1 and a_r=10 are almost equal in all cases. The sensitivity to a_r depends greatly on the initial position of the water table. The sensitivity is higher in the wet and average cases, and significantly lower in the dry case. The influence of f is also significant, with high values of f increasing the sensitivity to a_r.

Figures 3.20a and 3.20b represent the time evolution of runoff generation in the basin compared to the rainfall rate. Figure 3.21a and 3.21b represent the effect of the different runoff generation mechanisms on the total hydrograph obtained for the basin. These figures offer a more detailed description of the base case, represented in the second plot of figures 3.18a and 3.18b. The interpretation of the figures is analogous to the previous case.

Figures 3.20 and 3.21 show that the sensitivity of the total hydrograph to the anisotropy ratio comes almost exclusively from the return flow generated in permanently saturated areas (dotted line in the plots). The anisotropy ratio is the main cause driving lateral flow between elements, and therefore its main influence is through the effect of the subsurface flow on return flow. That explains why the model is so insensitive to the anisotropy ratio when R_i is interpreted as the initial recharge rate. In this case the initial moisture content is so low that most of the subsurface flow can be stored in the unsaturated area of the pixels without ever reaching the saturated areas. Only the case a_r=*500* generates a very small quantity of return flow.

In the case when R_i is interpreted as the initial moisture content, the model shows some sensitivity to a_r, specially in the case a_r=500. It can be observed that the portion of return flow generated in the areas which

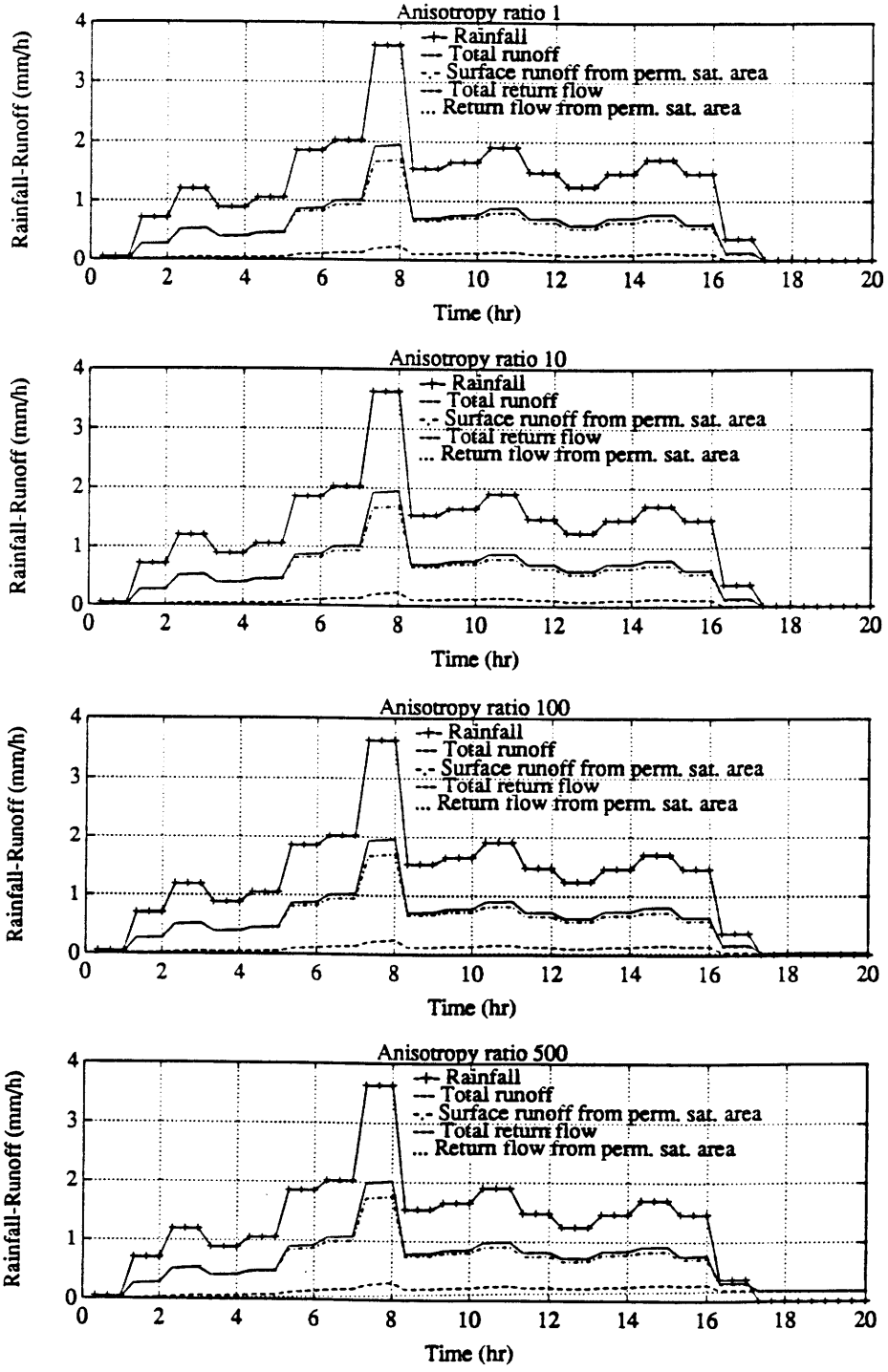


***Figure 3.20a**: Time evolution of different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content.*

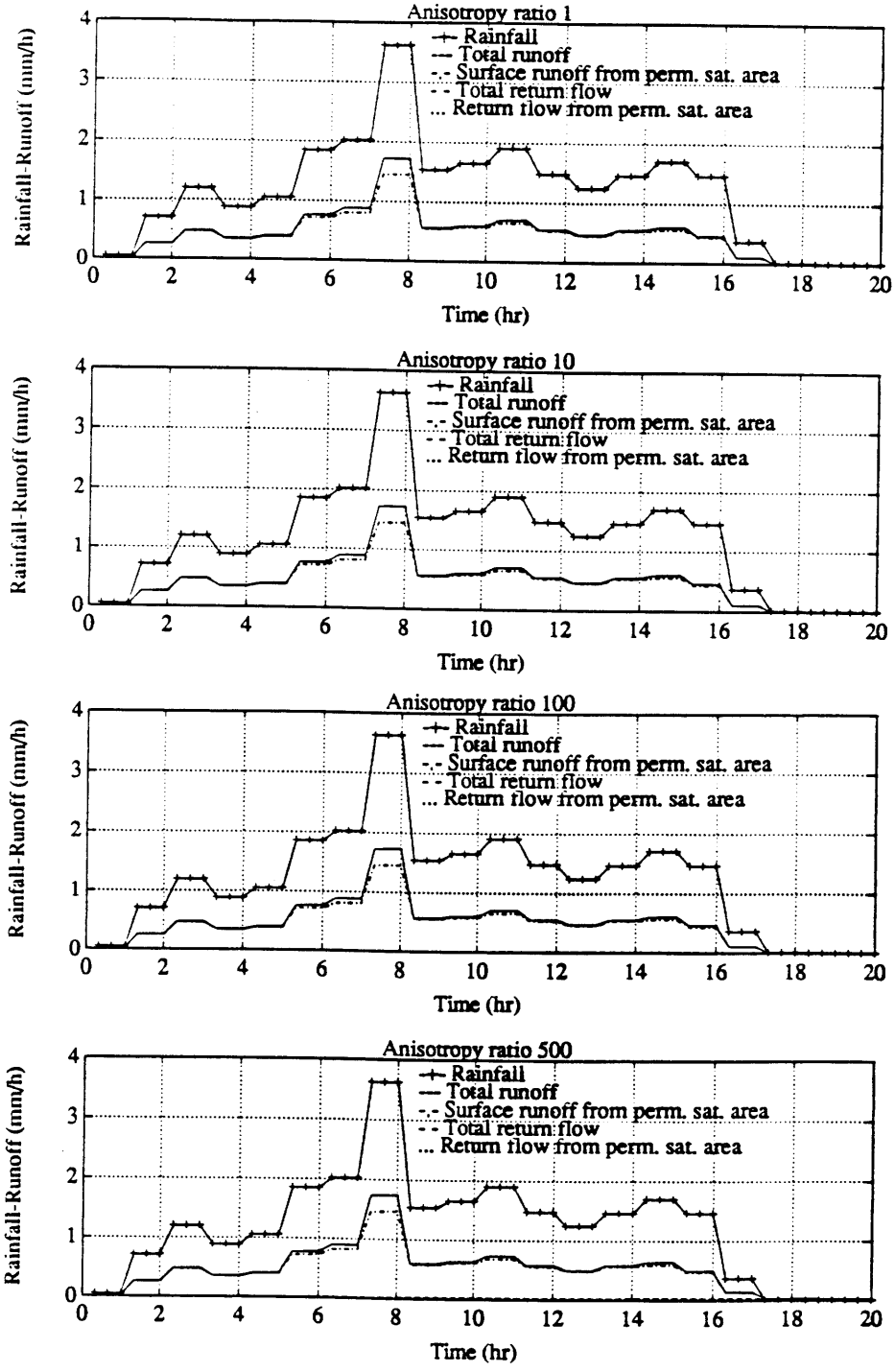


***Figure 3.20b**: Time evolution of different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

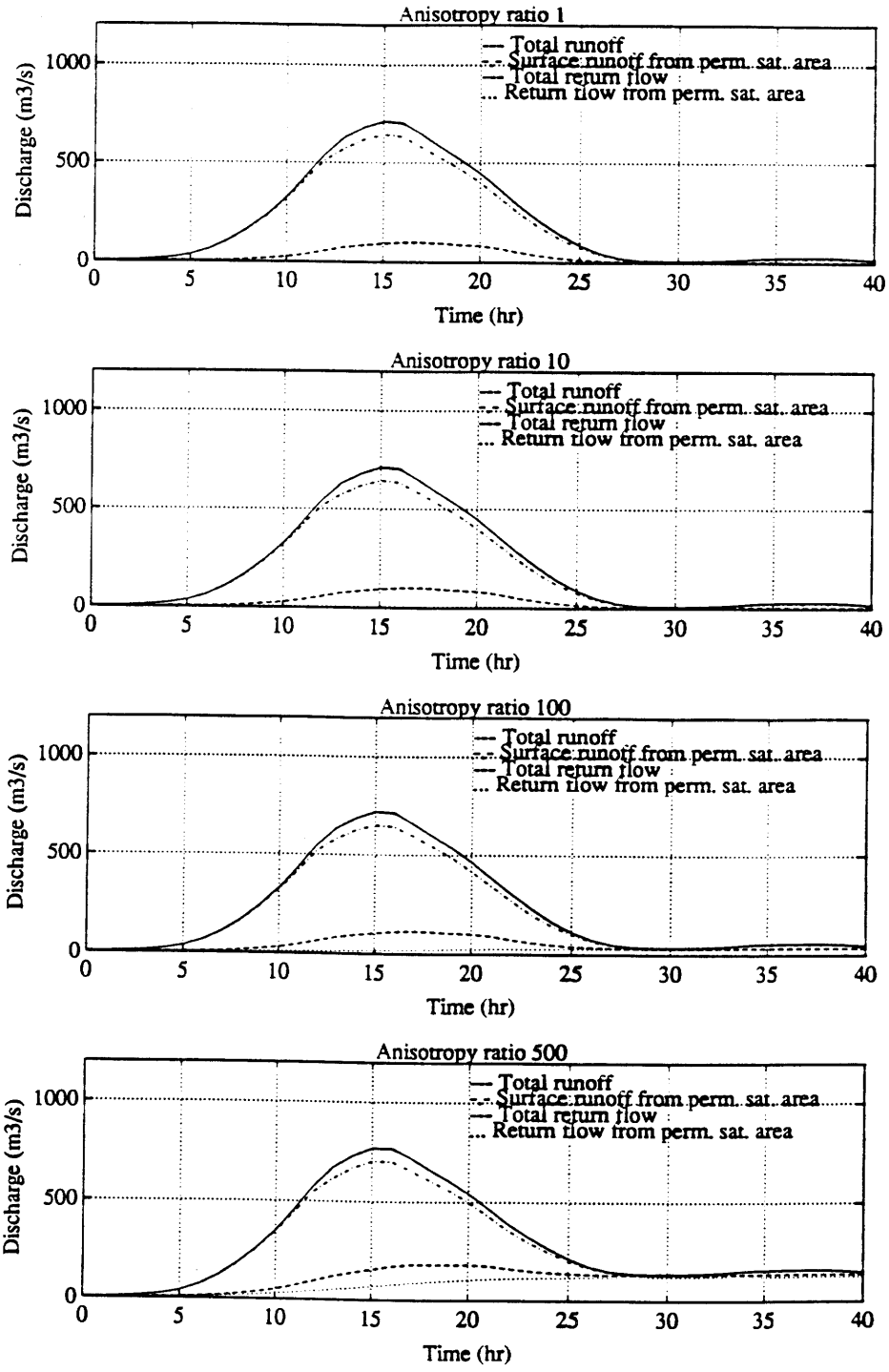


Figure 3.21a: *Decomposition of basin response into different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial moisture content.*

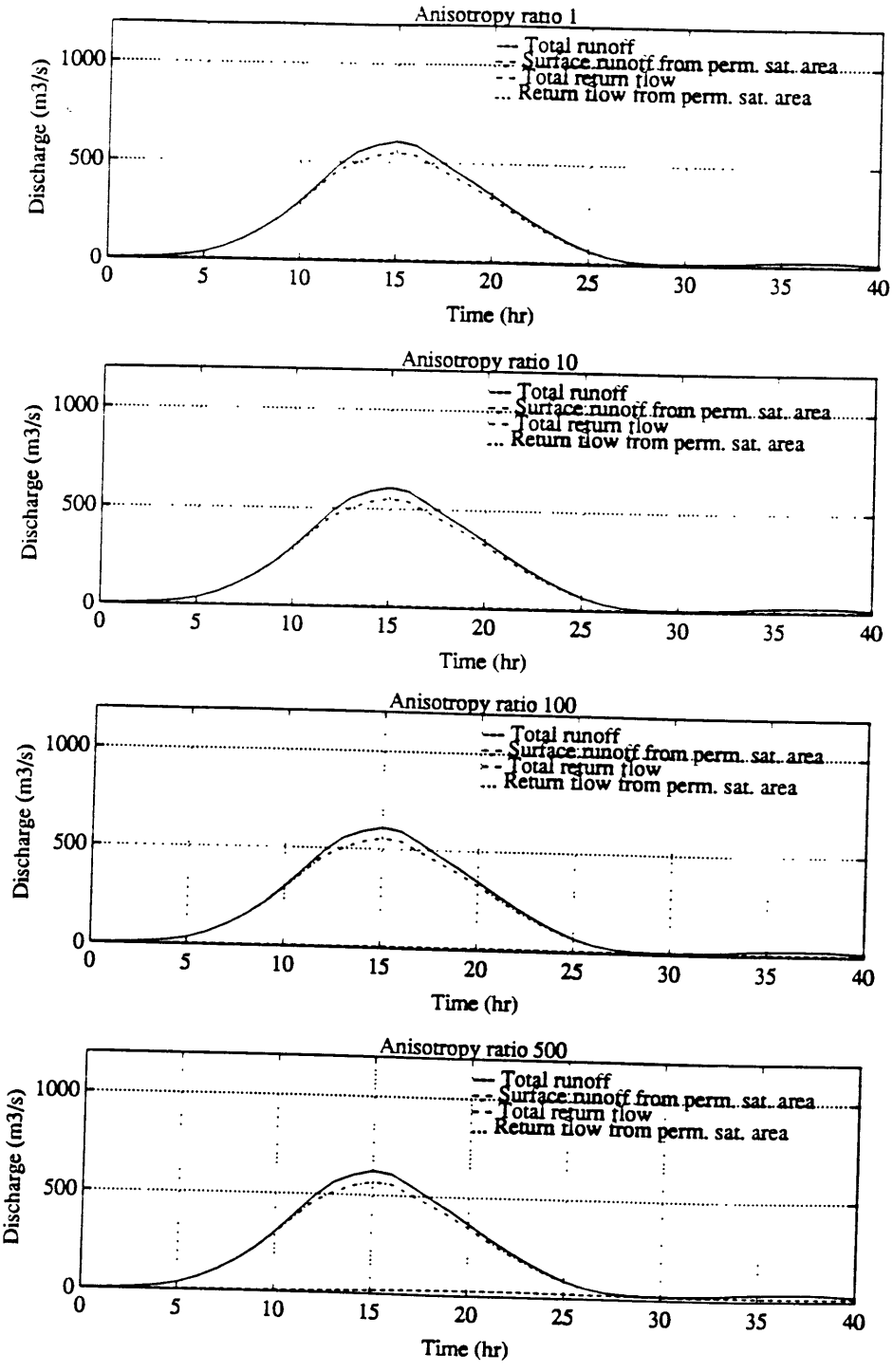


***Figure 3.21b**: Decomposition of basin response into different modes of runoff generation for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, initial state with 50% probability of exceedance and parameter R_i interpreted as initial recharge rate.*

become saturated during the storm is approximately constant for different anisotropy ratios. That dynamic return flow is mainly controlled by the parameter f, because local saturation is mostly a transient effect, and the response of the subsurface flow is so slow that there is no time to accumulate enough convergence of subsurface flow in the temporarily saturated areas. The effect of the anisotropy ratio can be seen in the areas permanently saturated because in that case there is enough time for the subsurface flow to accumulate.

The mechanism through which the anisotropy ratio increases runoff generation is by enhancing the production of return flow in permanently saturated areas. It is also clear that the response time of the runoff generated through this mechanism is much slower than other runoff components of the basin.

Sensitivity to the initial position of the water table

Figures 3.22 and 3.23 represent the sensitivity of the total hydrograph to the initial water table positions for different combinations of other model parameters. As in previous cases, the label 'a' corresponds to the interpretation of R_i as the initial moisture content and the label 'b', to the interpretation of R_i as the initial recharge rate. Figure 3.22 presents different values of the parameter f for an anisotropy ratio of 10, and Figure 3.23 presents different anisotropy ratios for a value of f=5 10^{-4} mm^{-1}.

The model is sensitive to the position of the water table in all cases, but it can be observed how different combinations of the other parameters can greatly enhance this sensitivity, specially when R_i is interpreted as the initial moisture content, because in this case the moisture content is

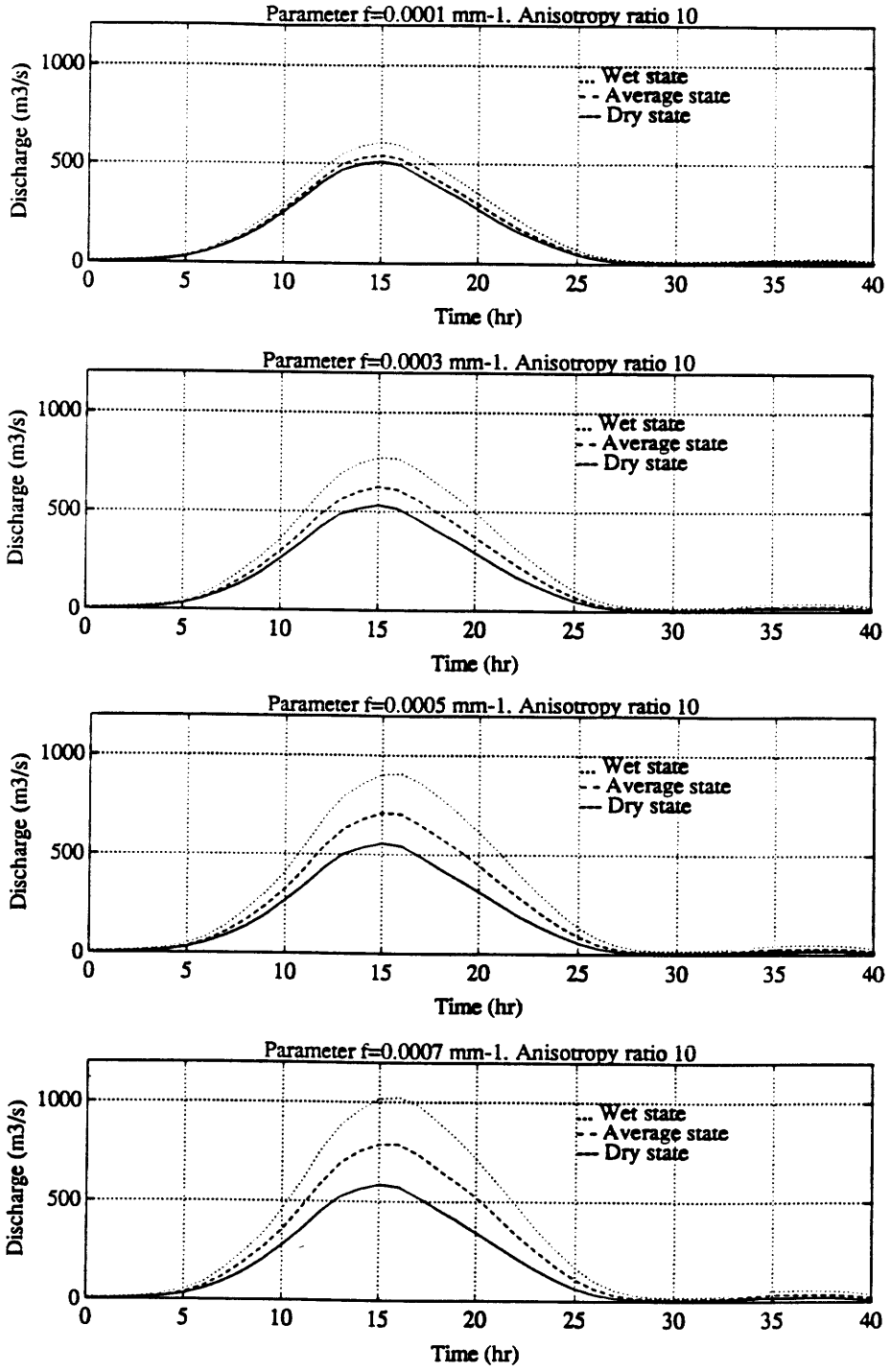


Figure 3.22a: *Sensitivity of basin response to initial state for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content.*

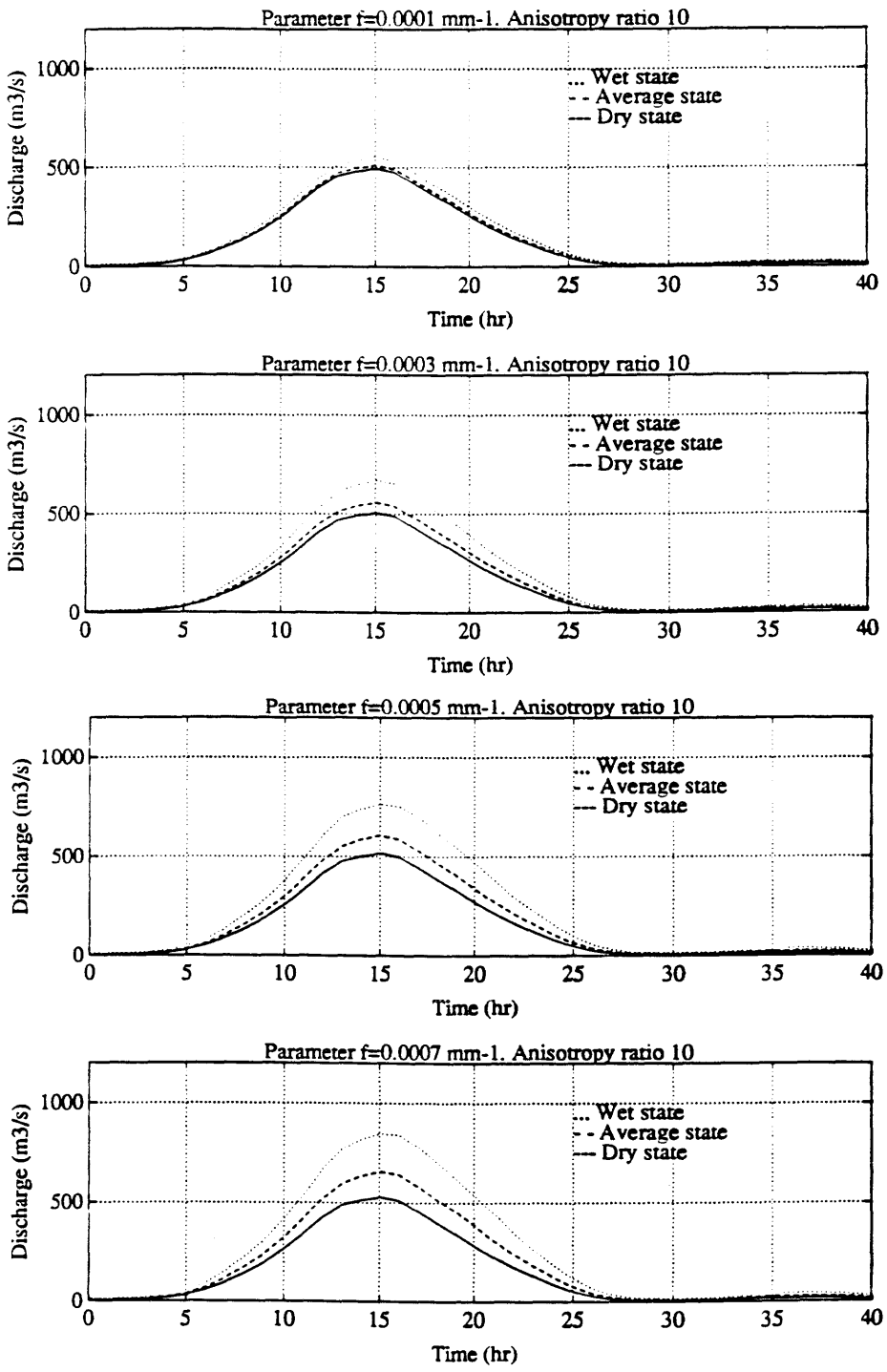


***Figure 3.22b**: Sensitivity of basin response to initial state for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate.*

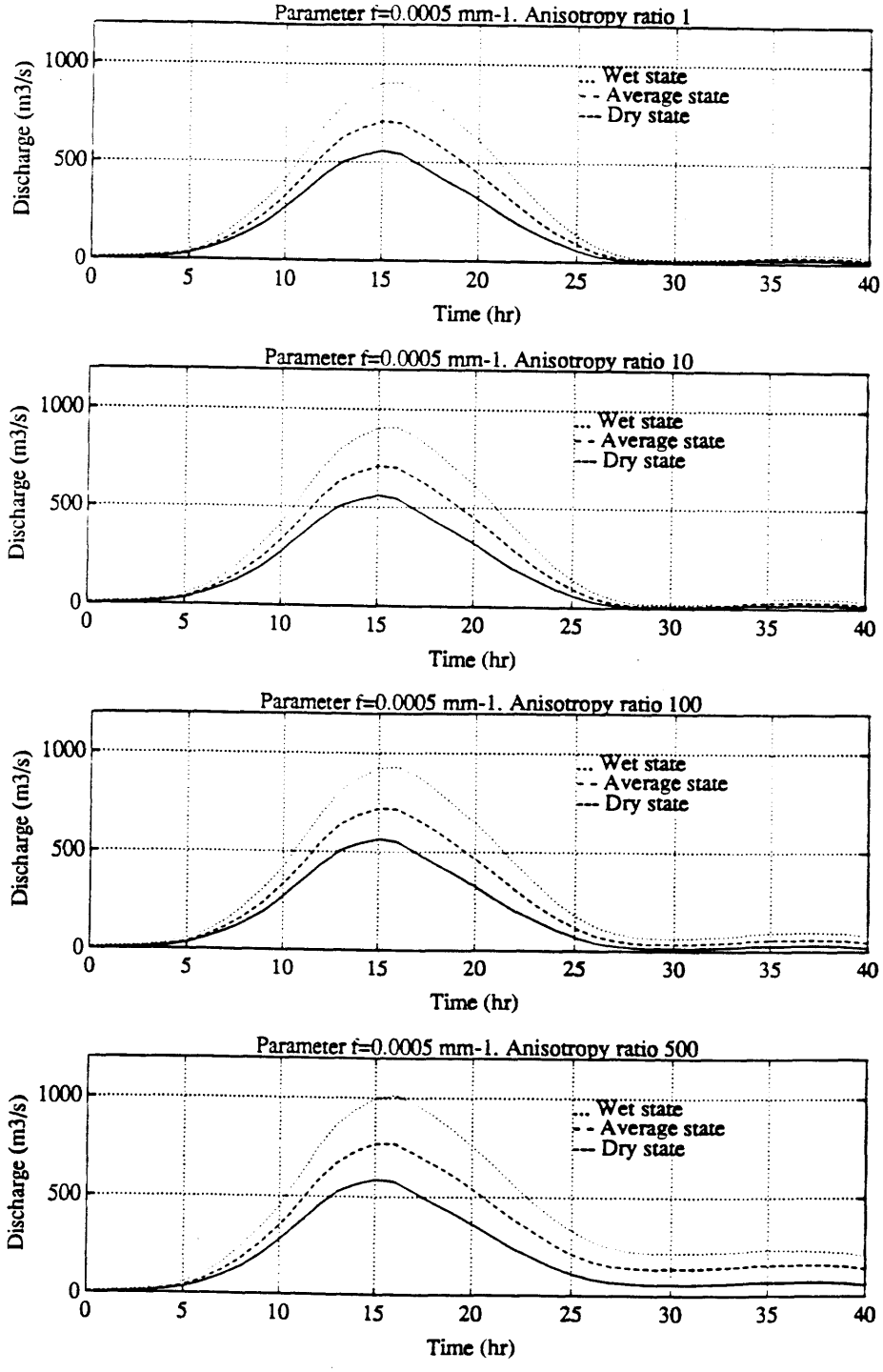


***Figure 3.23a**: Sensitivity of basin response to initial state for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and parameter R_i interpreted as initial moisture content.*

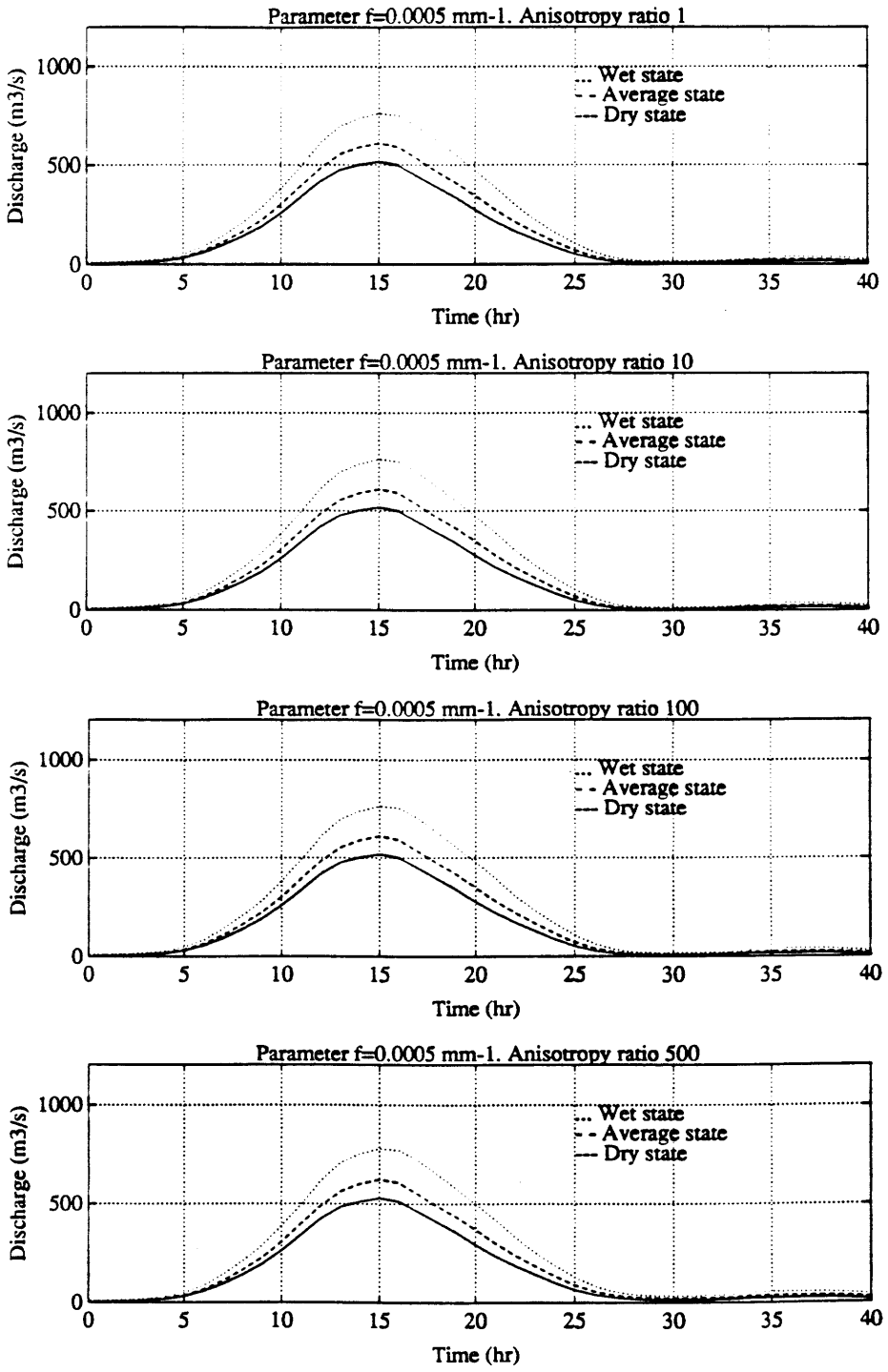


***Figure 3.23b**: Sensitivity of basin response to initial state for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and parameter R_i interpreted as initial recharge rate.*

correlated with the depth of the water table. In general, the effect of f on the sensitivity to the position of the water table is greater than that of the anisotropy ratio.

Figures 3.24a and 3.24b show the evolution in time of the runoff generation in the basin compared to the rainfall rate. Figures 3.25a and 3.25b represent the effect of the different runoff generation mechanisms on the total hydrograph obtained for the basin. The case considered in these figures is the one represented in the third plot of figures 3.22a and 3.22b, or in the second plot of figures 3.23a and 3.23b.

The sensitivity to the initial position of the water table is mainly a consequence of the surface runoff generated in the permanently saturated areas, which are those where the initial water table is at the surface. Different initial water table positions define different saturated areas, and therefore, they lead to different volumes of runoff. There is also an indirect effect on the generation of return flow, mostly as a consequence of different moisture contents in the case where R_i is interpreted as initial moisture.

The mechanism through which the initial water table position controls runoff generation is by defining the areas of the basin which are saturated. These areas have a runoff coefficient of one, and therefore their extension is extremely important to define the total runoff volume of the basin.

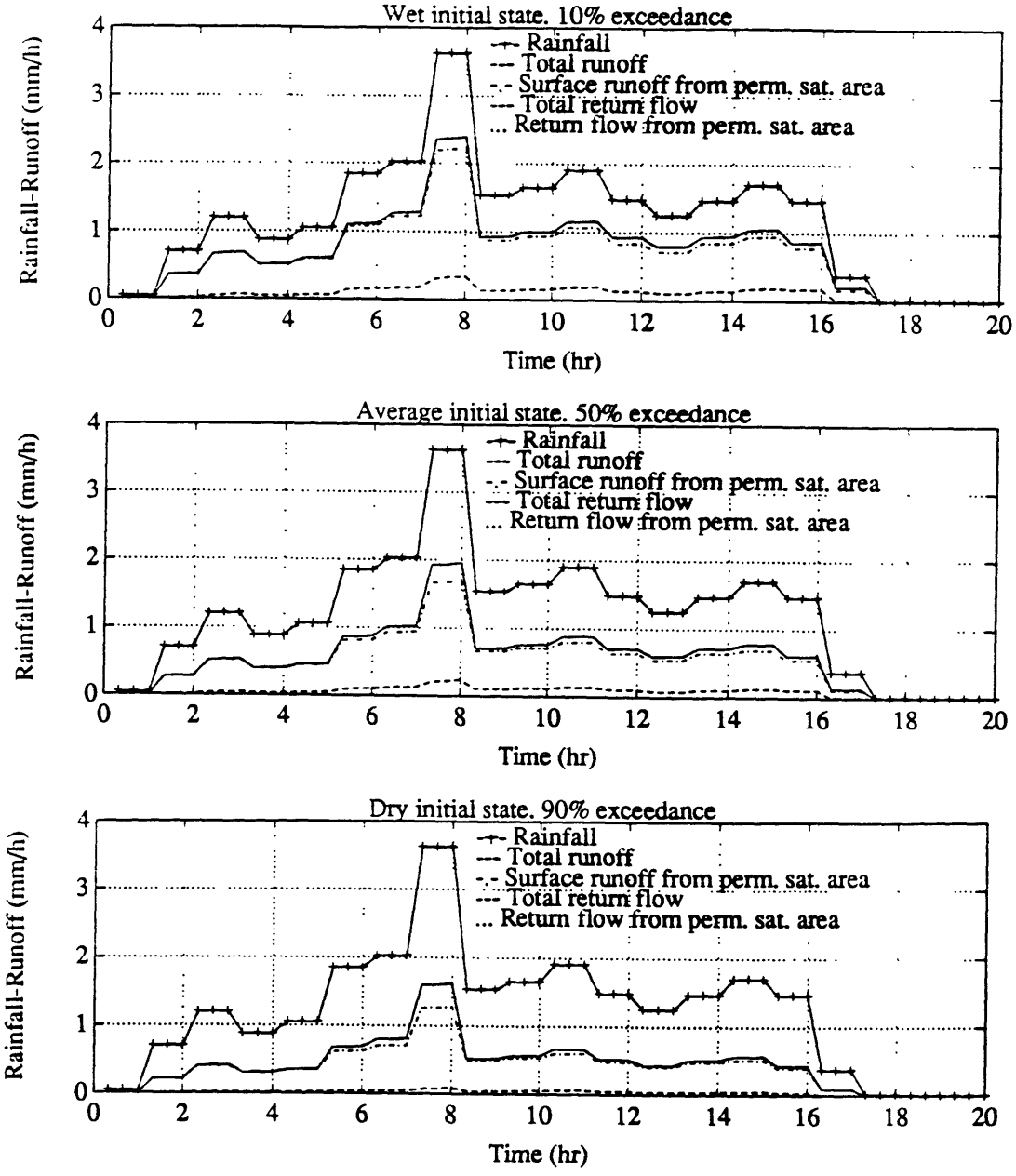


***Figure 3.24a**: Time evolution of different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content.*

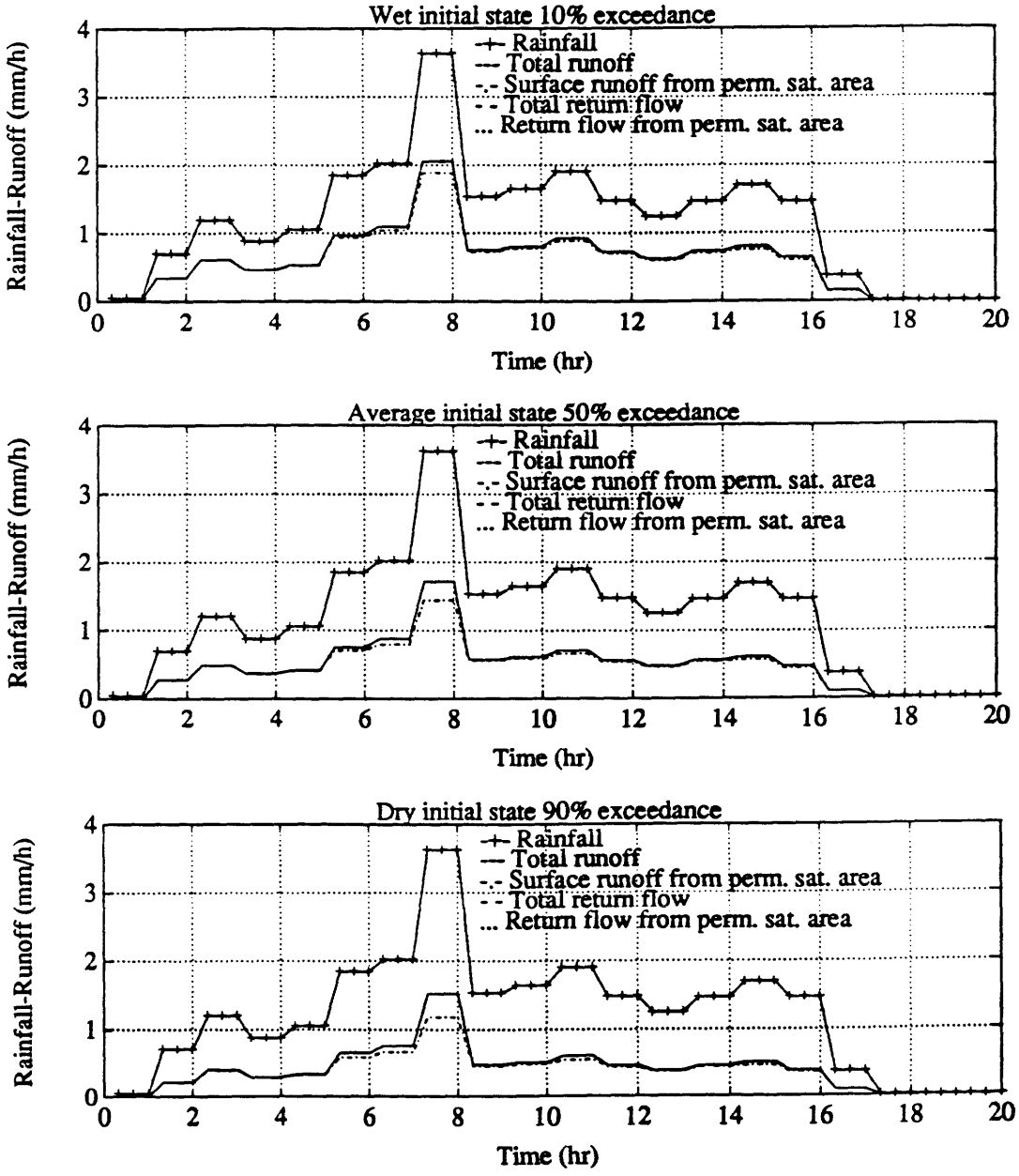


***Figure 3.24b**: Time evolution of different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate.*

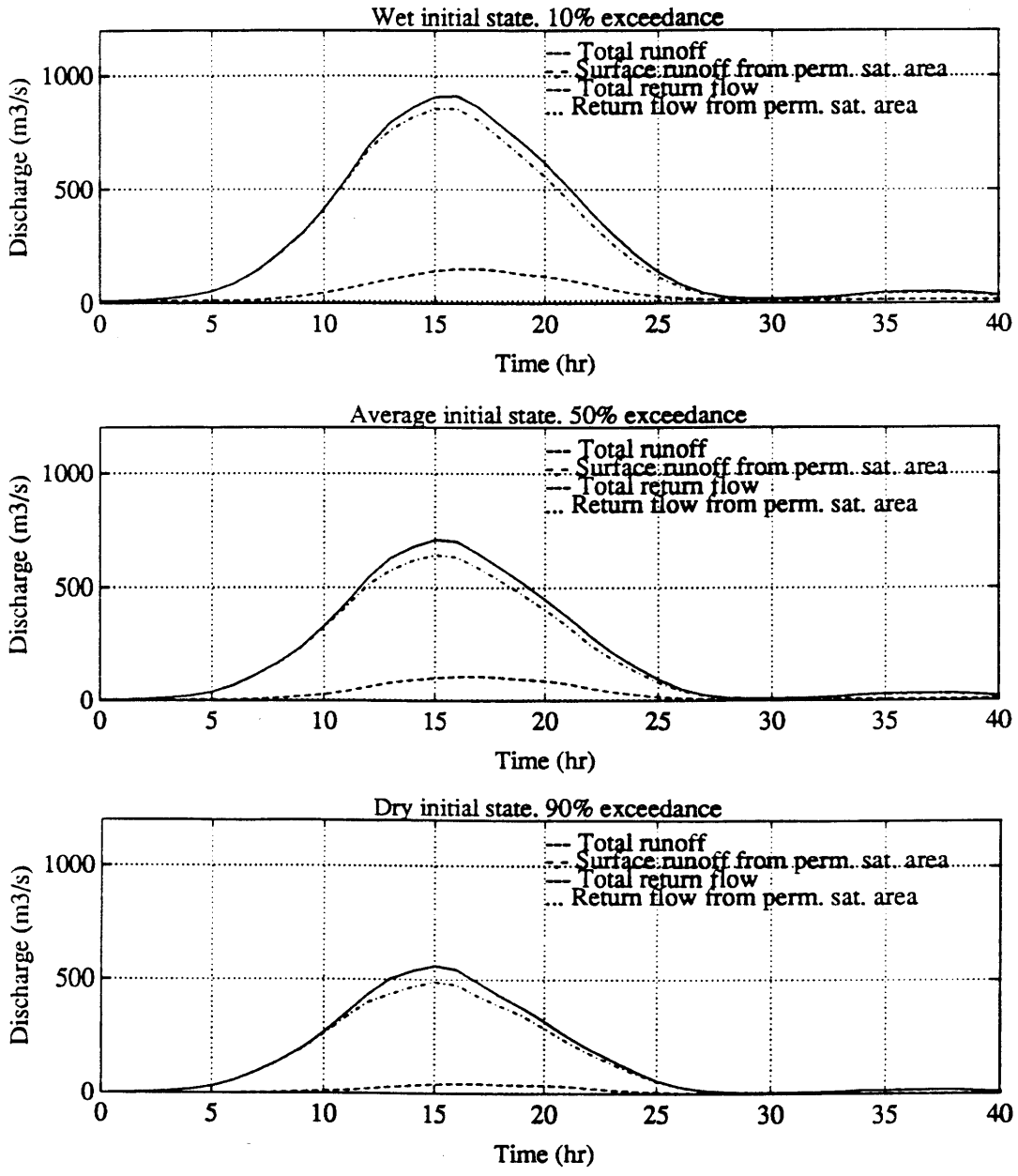


Figure 3.25a: *Decomposition of basin response into different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial moisture content.*

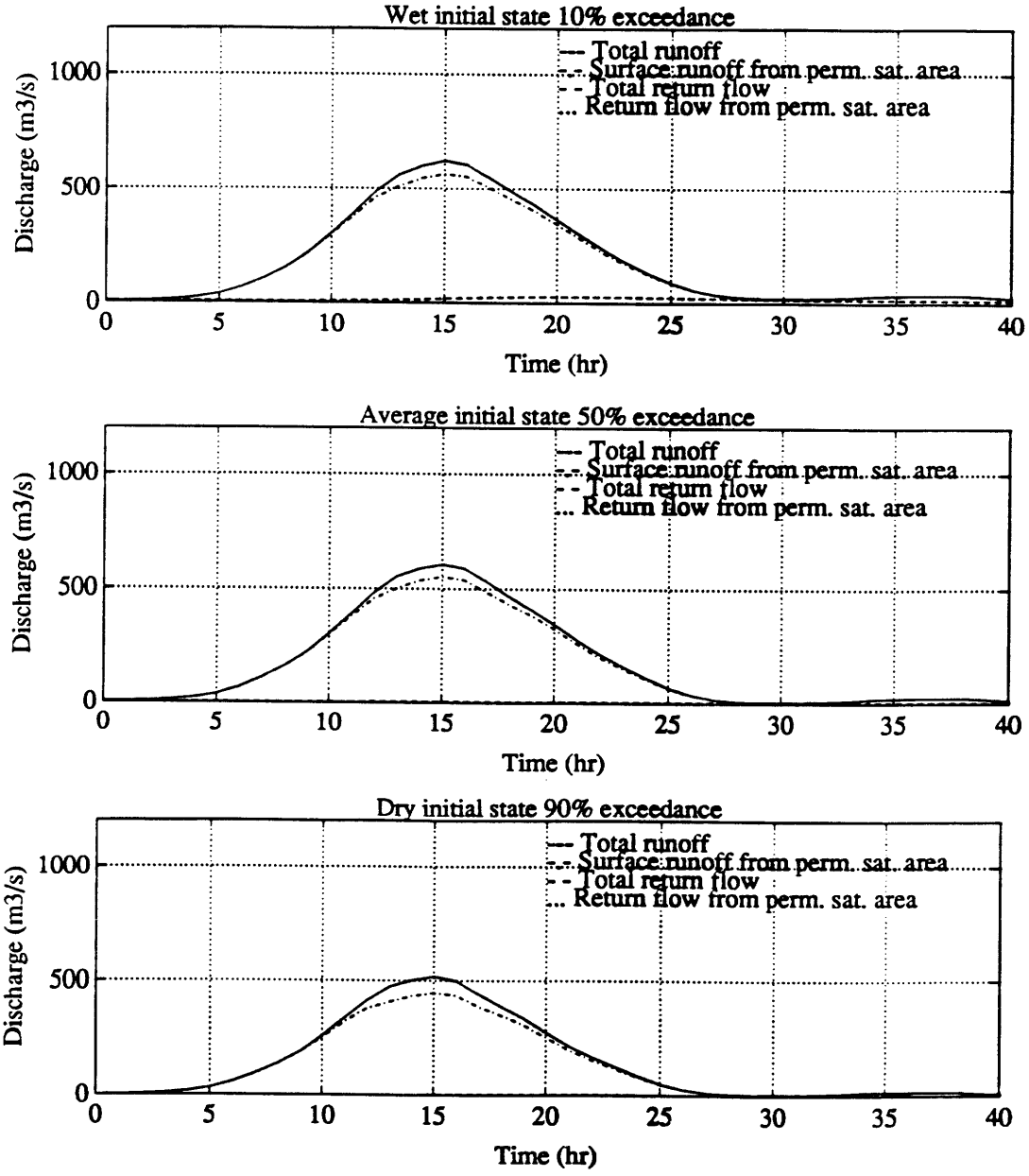


Figure 3.25b: *Decomposition of basin response into different modes of runoff generation for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and parameter R_i interpreted as initial recharge rate.*

Sensitivity to the interpretation of the parameter R_i

Figures 3.26 to 3.28 show the effect of the two different interpretations of the parameter R_i in the total hydrograph of the basin. Figure 3.26 shows the effect for different values of f, Figure 3.27 shows the effect for different values of the anisotropy ratio and Figure 3.28 shows the effect for different initial water table positions. The model is sensitive to the choice of R_i in all cases.

Figure 3.29 shows the time evolution of the generation of runoff compared to rainfall intensity. Figure 3.30 shows the effect of the different runoff generation mechanisms on the total hydrograph. The choice of R_i influences mostly the return flow generated in the areas of the basin which are not permanently saturated. When R_i is considered as the

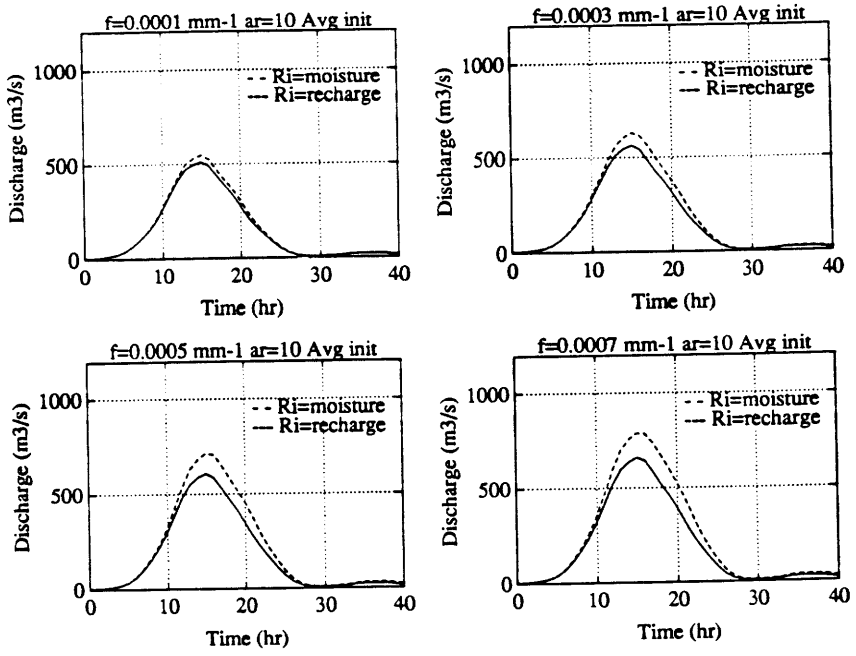


***Figure 3.26**: Sensitivity of basin response to the interpretation of parameter R_i for parameter f equal to 1 10^{-4}, 3 10^{-4}, 5 10^{-4} and 7 10^{-4} mm^{-1}. The results correspond to anisotropy ratio equal to 10 and initial state with 50% probability of exceedance.*

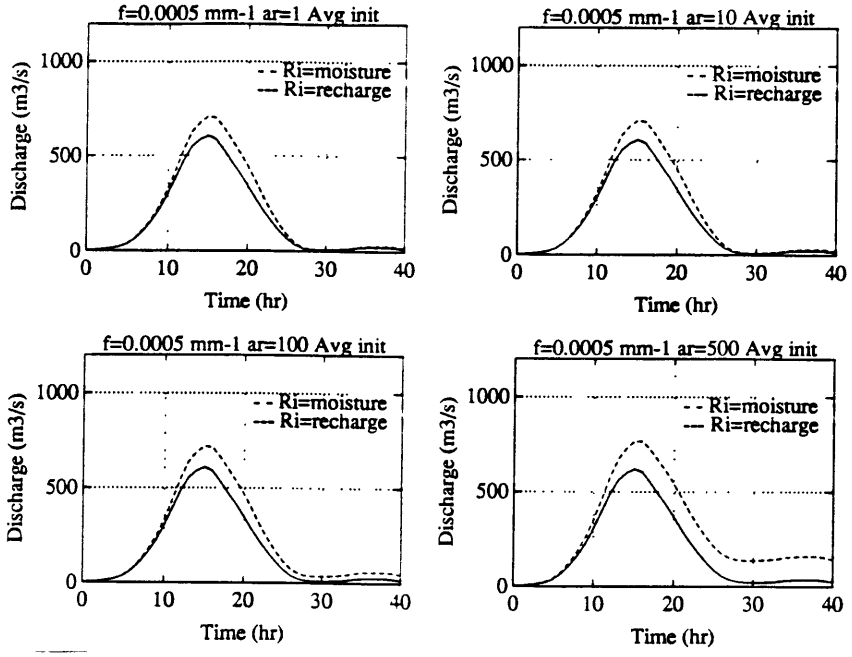


***Figure 3.27**: Sensitivity of basin response to the interpretation of parameter R_i for anisotropy ratio equal to 1, 10, 100 and 500. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and initial state with 50% probability of exceedance.*

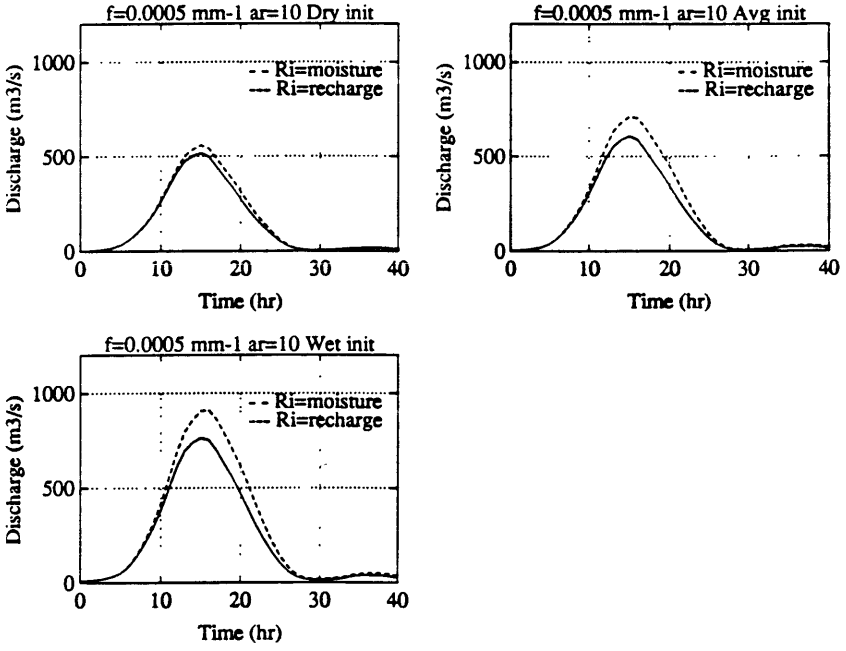


***Figure 3.28**: Sensitivity of basin response to the interpretation of parameter R_i for initial states with 10%, 50% and 90% probability of exceedance. The results correspond to parameter f equal to 5 10^{-4} mm^{-1} and anisotropy ratio equal to 10.*

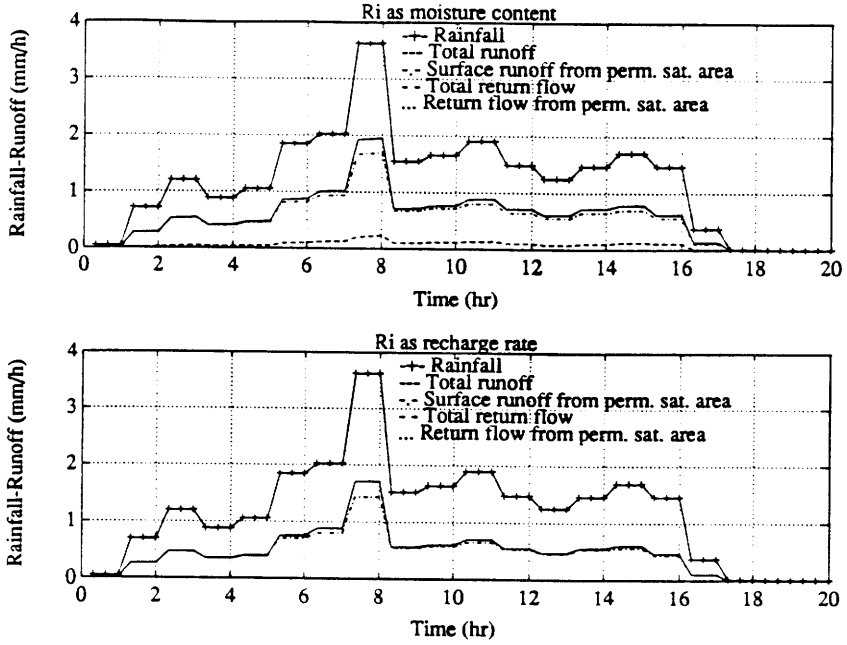


***Figure 3.29**: Time evolution of different modes of runoff generation for interpretation of parameter R_i as initial moisture content and as initial recharge rate. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and initial state with 50% probability of exceedance.*

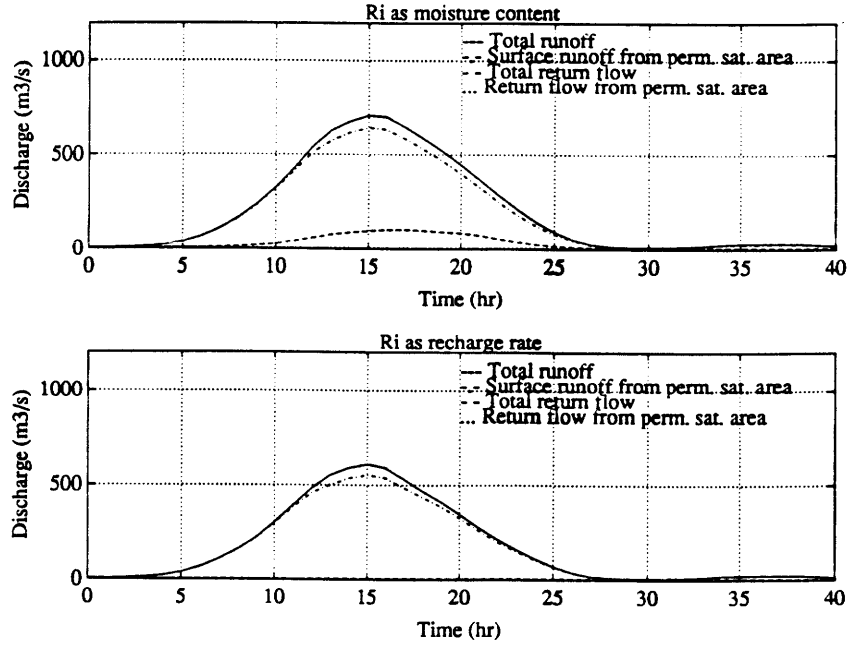


***Figure 3.30**: Decomposition of basin response into different modes of runoff generation for interpretation of parameter R_i as initial moisture content and as initial recharge rate. The results correspond to parameter f equal to 5 10^{-4} mm^{-1}, anisotropy ratio equal to 10 and initial state with 50% probability of exceedance.*

initial moisture content the areas where the water table is shallow are almost saturated. A small infiltration can saturate them, and these areas generate return flow. When R_i is considered as the initial recharge rate the moisture content above the water table is homogeneous throughout the basin, and it takes longer to saturate significant portions of the basin.

Sensitivity to surface routing parameters

A different set of computer runs was used to test sensitivity to surface routing parameters. The model was run with a unique set of runoff-generation parameters (the base case), and the surface-routing parameters were changed to explore model sensitivity. The parameters tested were the ratio of stream velocity to hillslope velocity, K_v, and the coefficient c_v and exponent r in the power law relating hillslope velocity and discharge at the outlet (Equation 2.46). The numerical values of the parameters are shown in Table 3.4.

Table 3.4 *Parameter values for the sensitivity to surface routing*

Parameter				
Velocity ratio K_v	5	10	15	20
Coefficient c_v (m h^{-1})	100	200	400	600
Exponent r	0.0	0.1	0.2	0.3

Figures 3.31 to 3.33 show the results of the sensitivity analysis to routing parameters. Figure 3.31 shows sensitivity to the ratio of velocities K_v, Figure 3.32 shows sensitivity to the coefficient c_v, and Figure 3.33 shows sensitivity to the exponent r. Sensitivity is very large in all cases,

and, although total hydrograph volume is conserved in all plots, the shapes are greatly affected by the routing parameters. The effects of K_v and c_v are very similar, because both represent uniform increments in travel velocities. In the case of K_v, the increments only correspond to stream velocities, while in the case of c_v, the increments correspond to both stream and hillslope velocities. In both cases the effect of a decrease in the parameter is the dampening of the basin response, delaying the occurrence of the peak and smoothing the shape of the hydrograph. It also appears that stream velocities are dominant in configuring basin response, at least for the range of parameter values tested here.

The effect of the exponent r is somewhat different than those of K_v or c_v. Increments of the exponent r produce a stronger narrowing of the shape of the hydrograph than increments of the other parameters do. The

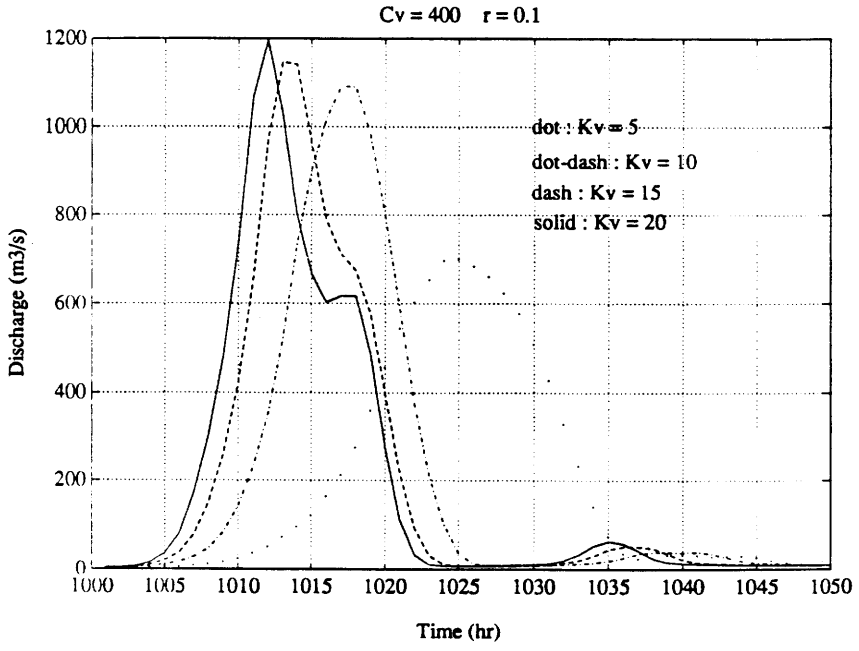


***Figure 3.31**: Sensitivity of basin response to the ratio of stream and hillslope velocities K_v. The results correspond to coefficient c_v equal to 400 mh^{-1} and exponent r equal to 0.1.*

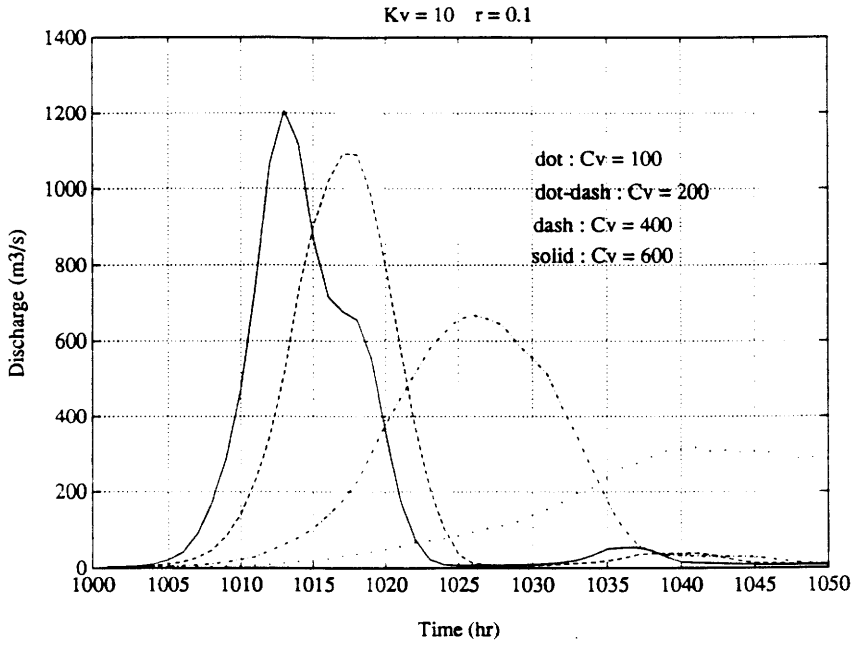


***Figure 3.32**: Sensitivity of basin response to coefficient c_v. The results correspond to velocity ratio K_v equal to 10 and exponent r equal to 0.1.*

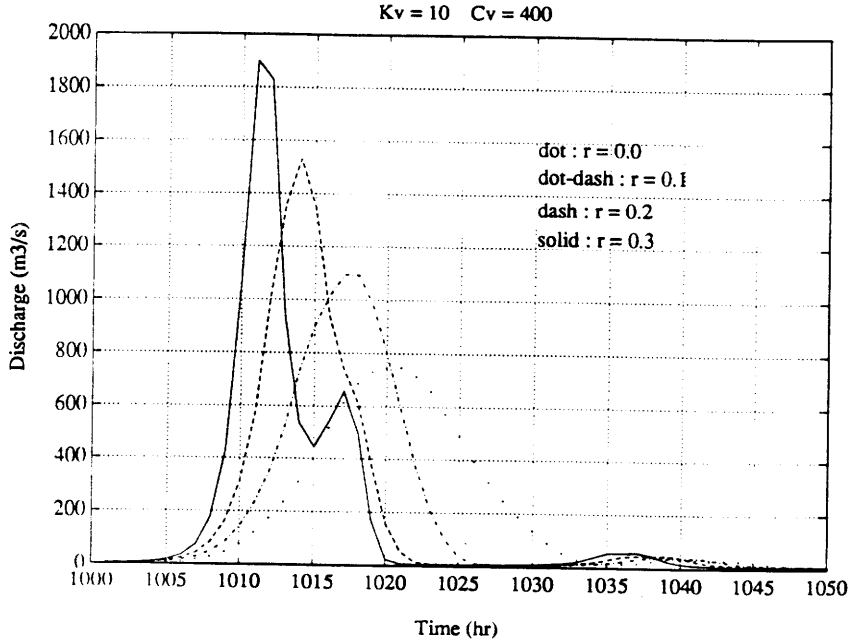


***Figure 3.33**: Sensitivity of basin response to exponent r. The results correspond to velocity ratio Kv equal to 10 and coefficient c_v equal to 400 mh^{-1}.*

explanation is that increments in r represent greater increments of travel velocities when the discharge is high, and therefore, the velocities are relatively higher around the peak and lower at the beginning and the end of the hydrograph. This should be the basic criterion to identify non-linearities in basin response.

3.2 Model calibration and evaluation

The ability of the model to reproduce observed stormflow is analyzed in this section. The data available are described first, together with the suggested methodology for model calibration. The original data were divided in two groups, one for calibration and another for evaluation. The results of the calibration process are shown in Section 3.2.2, and the performance of the model with the evaluation set is analyzed in Section 3.3.3.

3.2.1 Calibration methodology

The model is applied to the Sieve basin, already described in Section 3.1.2. Data for the calibration are taken from Cabral (1990), which applied a preliminary version of DBS to the Sieve basin. She collected data about physical parameters and several observed storms.

Rainfall data at a temporal resolution of 20 minutes are available from a number of recording stations in the Sieve basin area, shown in Figure 3.11. Up to 1984, the only recording station located inside the basin was Borgo San Lorenzo. On a Thiessen polygon analysis, Borgo San

Lorenzo covers approximately 75% of the basin area, and other three stations (Firenze-Ximeniano, Camaldoli and Vallombrosa) cover the remaining portions. Increasingly higher spatial resolution is available for more recent storms (since 1985), but no radar data are available yet.

Streamflow data are available from a stage gauge located at Fornacina. The gauge does not operate continuously; it only functions during storm periods which are considered to be potentially dangerous. Stage readings are available on an hourly basis, and only cover a few hours of each storm. No inter-storm flow data are available. A rating curve is used to translate stage readings into streamflow data.

A total of ten storm events were selected for the calibration and evaluation processes. The selection was made based on the simultaneous availability of rainfall and streamflow information. Five storms were selected for the calibration step, reserving other five storms for the evaluation step. The storms included in the calibration process are summarized in Table 3.5.

***Table 3.5** Storms for calibration*

Storm	**Total rainfall depth (mm)**
February 1977	33.04
January 1979	46.36
November 1982	74.45
February 1983	38.34
January 1985	25.88

As a conclusion of the sensitivity analysis, the following calibration methodology was adopted. There are two processes that have to be reproduced by the model: how much runoff is generated during the storm

and how that runoff is transported through the drainage network to the basin outlet. Athough they are obviously interconnected, because the amount of water available affects flow velocity, the simplification of analyzing both independently is made. Model structure supports this strategy, since runoff generation and flow routing are two independent steps in model inference. Of course, the strategy will work better for cases where non-linearities in basin response are small. A second, fine-tunning iteration can be carried out if flow velocity is found to be strongly dependent on runoff volume.

Calibration of runoff generation

The first issue to address while calibrating runoff generation is defining a well-posed problem. The model has a very large number of parameters distributed over the basin, and observations are only available for a few locations. Therefore, a number of a priori assumptions have to be made about model parameters in order to keep the model parsimonious. The goal is to fit as many model parameters as possible using means other than calibration, and leaving only a few key parameters for the calibration step. According to the sensitivity analyses, three model parameters are crucial in determining surface runoff: surface hydraulic conductivity, K_{0n}, its rate of decrease with depth, f, and anisotropy ratio, a_r. Original data are contained in the DEM and in the soil study. The soil study available for the Sieve basin provided information about the hydraulic conductivity and porosity for each of 17 soil types identified in the basin. The fact that hydraulic conductivity data are available simplifies the calibration process, because it substracts one degree of

freedom. Only two model parameters, f and a_r, were left to be estimated through calibration. Other model parameters are apparently less important, and were fit by means other than direct calibration.

Normal hydraulic conductivity values were assigned to every soil type according to the values proposed in the soil study, which were assumed to represent vertical infiltration. However, the original data were uncertain, with large variabilities within a given soil type, and the initial solution of considering the mean value for each soil type was only preliminary, to be revised during calibration if unreasonable results were obtained. Porosity was also available in the soil study, but other soil properties of the Brooks-Corey parameterization had to be estimated from published values in the literature for soil with analogous conductivity and texture. Table 3.6 shows the original data contained in the soil study and the values adopted for model parameters of each soil type.

Table 3.6*Original data in the soil study and values adopted in the model (from Cabral et al., 1990)*

Soil type	**Soil texture**	**Estimated K (mm h^{-1})**	**K_{0n} (mm hr^{-1})**	θ_s	θ_r	ε
Type 1	FS-FLA	2-41	21.5	0.53	0.02	3.6
Type 2	FL-FLA	0.2-7	3.6	0.52	0.036	3.6
Type 3	A	0.25	0.01-0.5	0.48	0.09	7.5
Type 4	A	0.25	0.01-0.5	0.48	0.09	7.5
Type 5	FA	45.	20-70	0.56	0.109	3.6
Type 6	A	25.5	1-50	0.56	0.09	7.5
Type 7	FL-FLA	0.2-7	3.6	0.53	0.109	3.6
Type 8	FL	5.1	0.2-10	0.49	0.109	3.6
Type 9	F-FA	16.6	0.2-33	0.52	0.064	3.5
Type 10	FS	21.8	2.7-41	0.48	0.036	3.4
Type 11	FS-FA	0.2-41	20.6	0.52	0.072	3.6
Type 12	FS-FA	0.2-41	20.6	0.52	0.072	3.6
Type 13	FS-FA	0.2-41	20.6	0.52	0.072	3.6
Type 14	F-FS-FA-FL	0.2-41	20.6	0.50	0.07	3.6
Type 15	F-FS	21.8	2.7-41	0.49	0.03	3.5
Type 16	FS	21.8	2.7-41	0.48	0.041	3.6
Type 17	Detritic	40	40.	0.25	0.02	3.4

The calibration problem was then posed as obtaining reasonable values for f and a_r that produced the best collective agreement between computed and observed streamflow for the events reserved for calibration. During the sensitivity analyses, f and a_r were found to be relatively independent. f controls the volume of infiltration-excess runoff and a_r controls the volume of subsurface runoff. The difference is observable, since infiltration-excess runoff is directly related to rainfall, whereas subsurface runoff may occur after precipitation has finished. Therefore, the tail of the observed hydrograph can be used to obtain an estimate of short-term subsurface runoff tuning the value of a_r. The other component of runoff is then estimated tuning the value of f. The process required a thorough analysis of every storm available, trying to discriminate between direct and subsurface runoff, and was based on qualitative hydrologic judgement rather than on mathematical optimization of an objective function. It was also complicated by the fact that the observations of runoff were usually discontinued when water levels returned to normal values, and therefore, only a small fraction of the tails of the hydrographs was available.

Another important issue had to be addressed in the calibration of runoff volume: the initial state. Basin initial state is characterized by the position of the water table at every grid point and the initial moisture content in the soil column, represented by R_i. In principle, the initial state of the basin is either estimated based on remote sensing of surface moisture distribution or computed with a long-term hydroclimatological model that includes evapotranspiration. In practice, the only data available for the Sieve basin were precipitation series at the recording stations, but no other climatological measurements were taken.

The estimation of the initial state based exclusively on precipitation is too simplistic, since evapotranspiration is also an important factor. Figure 3.34 and Table 3.7 show an analysis of this aspect for the Sieve data set. Figure 3.34 shows the rainfall registered by the Borgo San Lorenzo gauge during the 100 days prior to every storm event. Rainfall is presented as cumulative values preceding the event, that is, with time running backwards since the beginning of the storm. A good measure of the antecedent moisture condition is the global runoff coefficient of the event, although other factors, such as total volume and distribution of rainfall also play a role. As Table 3.7 shows, there is no apparent consistent relation between runoff coefficient and antecedent precipitation at different time horizons for all five storms. The variability cannot be explained only by total precipitation volume or average rainfall intensity,

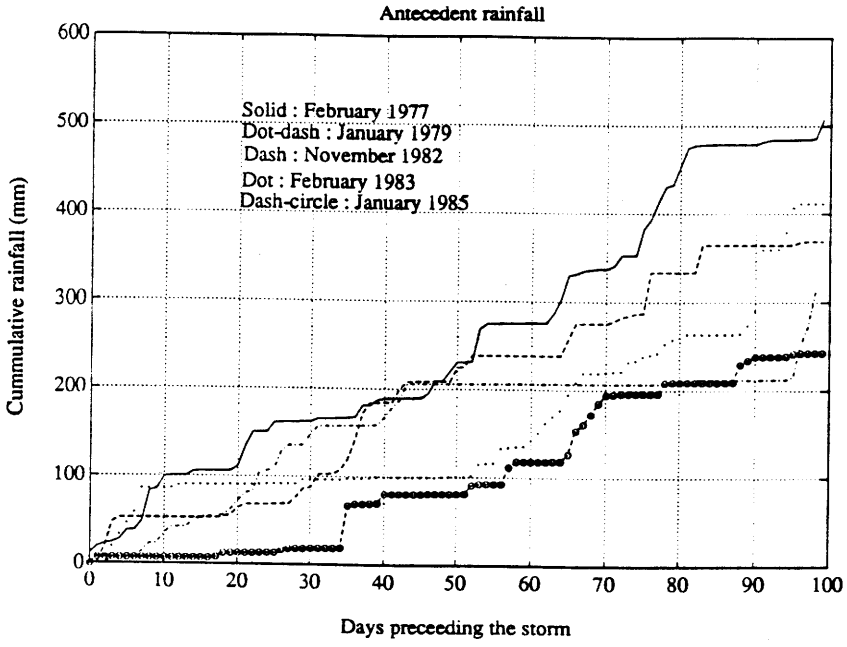


Figure 3.34: *Antecedent rainfall corresponding to the storms in the calibration set. The plot shows cummulative rainfall in mm preceeding the storm as a function of time in days.*

and we can conclude that the antecedent moisture condition is not a function of antecedent precipitation alone. No measurements were available to estimate the influence of other variables, and therefore no independent estimate of the initial state of the basin was available.

Table 3.7 *Cumulative antecedent precipitation (mm) for the storms in the calibration set*

Storm	1 day	2 days	1 week	2 weeks	30 days	Runoff Coeff.
February 1977	11	19	38	101	162	0.90
January 1979	0	0	7	41	135	0.85
November 1982	0	0	52	52	86	0.50
February 1983	7	9	59	90	97	0.80
January 1985	0	7	7	7	18	0.75

The solution adopted to deal with the problem of the initial state of the basin was to leave it as another unknown to be estimated by calibration. We make the initial state a function of just one variable, the uniform recharge rate R_i that is in long-term equilibrium with observed interstorm flows in the basin. Long-term records of streamflow in the Arno river enable us to estimate the distribution of monthly averages of inter-storm flow in the Sieve. For every month, a value of inter-storm streamflow can be assigned to a given probability of exceedance. Flows with low probability of exceedance represent wet states and flows with high probability of exceedance represent dry states. For every month three probability levels were selected: 0.1, 0.5 and 0.9, and the corresponding inter-storm flows were obtained (see Cabral, 1990).

Inter-storm flows were assumed to be in equilibrium with a uniform recharge rate in the basin, and the water table evolution model developed by Cabral (1990) was used to obtain the distribution of water table depths in the basin. According to the sensitivity analyses, the parameter R_i was

interpreted as initial moisture content of every pixel. Three initial states were thus defined for every month, and calibration was carried out in parallel for all three states of the corresponding month in every storm. This weakened considerably the calibration process, since a lot of the variability in basin response can be explained through different initial conditions, but no other independent means of estimating the initial state were available.

Calibration of time of travel

The scheme followed in the calibration of the time of travel is much simpler, since the hypothesis of constant velocity throughout the basin reduces the complexity of the problem. The drainage network was defined independently of the calibration, since there are physical bases to estimate it. The network proposed by Cabral et al., (1990), based on studies of Carla et al. (1986), was adopted. The drainage network is generated considering a threshold contributing area of 8 elements, which corresponds to 1.28 km^2. The resulting network has 1084 stream elements out of a total of 5252 for the whole basin.

Travel velocities are given by Equations (2.46) and (2.47). The parameters to estimate are those of Equation (2.46), the coefficient c_v and the exponent r, and the ratio of stream velocity to hillslope velocity, K_v. The estimation of c_v and K_v was based on the comparison of the general features of the hydrograph, notably the time at which the main peak occurs.

3.2.2 Results of the calibration process

A trial and error process was followed during calibration. Once satisfactory values for the routing parameters were obtained, different combinations of f and a_r were tried until acceptable results were obtained. The quality of the original data did not support a mathematical parameter optimization, and the objective of the calibration was only to obtain a rough estimate of the approximate values of the parameters. Therefore, a_r and f were taken from a discrete domain.

Table 3.8Parameter set which gave the best fit

Parameter	Value
Parameter f (mm^{-1})	7×10^{-4}
Anisotropy ratio a_r	500
Velocity ratio K_v	12.75
Velocity coeffcient c_v Kmh^{-1}	5.2
Exponent r	0

All storms were simulated for the three initial conditions: dry (90% probability of exceedance), average (50% probability of exceedance) and wet (10% probability of exceedance). The evaluation of the results was made considering all five storms simultaneously. The selection of the best parameter set was made based on the comparison of the observed data with the simulations corresponding to the three initial conditions. The objective was to reproduce basic features of the observed flow, such as peak discharge, time to peak and baseflow recession. Since the initial state was not known, there was one degree of freedom in the results. The evaluation of the results was obviously subjective, and the assumption

was made that the storms available for calibration should be comprised between the two limit initial conditions, 10% and 90% probability of exceedance respectively. The parameter set which gave the best fit is listed in Table 3.8.

Results obtained for what we consider the best fit are presented in Figures 3.35 through 3.39. General characteristics of the basin response are well captured by the model, but the fit between observed and computed hydrographs is less than optimal, specially considering that the initial state is a variable of the calibration process. However, the results can be evaluated as acceptable, given the large uncertainty about the real rainfall (75% of a basin of 840 km^2 is covered by just one raingauge in most cases) and the sporadic nature of the streamflow observations.

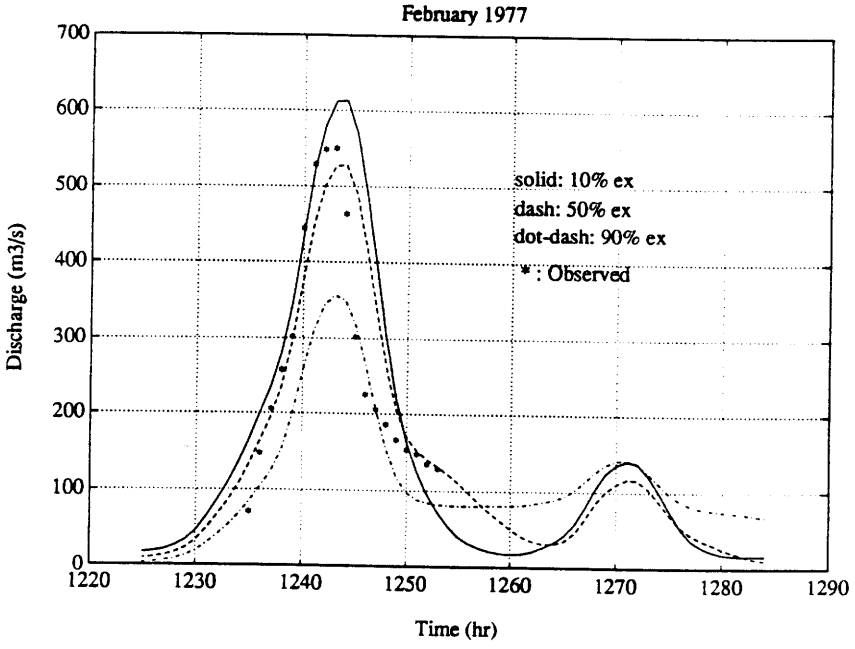


***Figure 3.35**: Observed and simulated hydrographs for the storm of February 1977. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

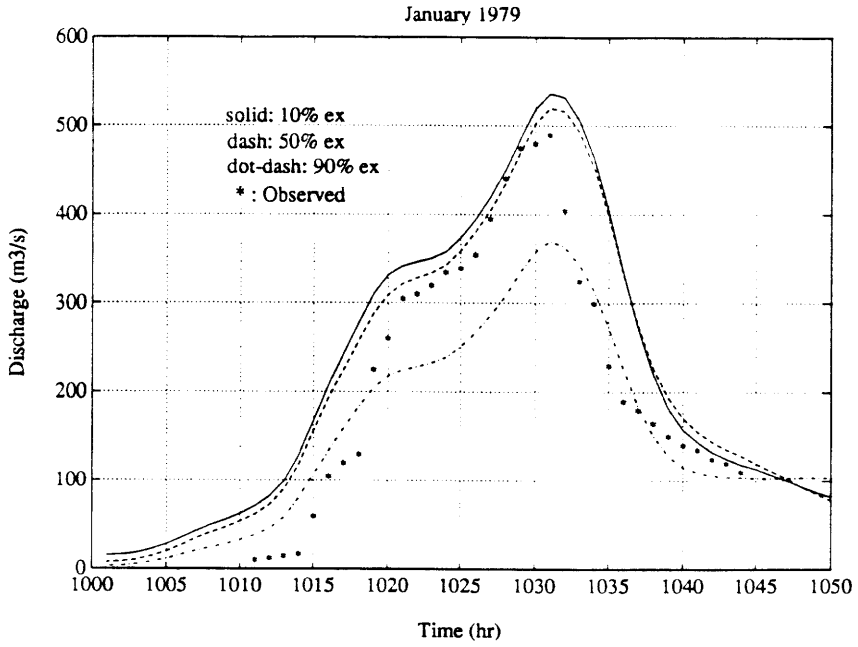


***Figure 3.36**: Observed and simulated hydrographs for the storm of January 1979. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

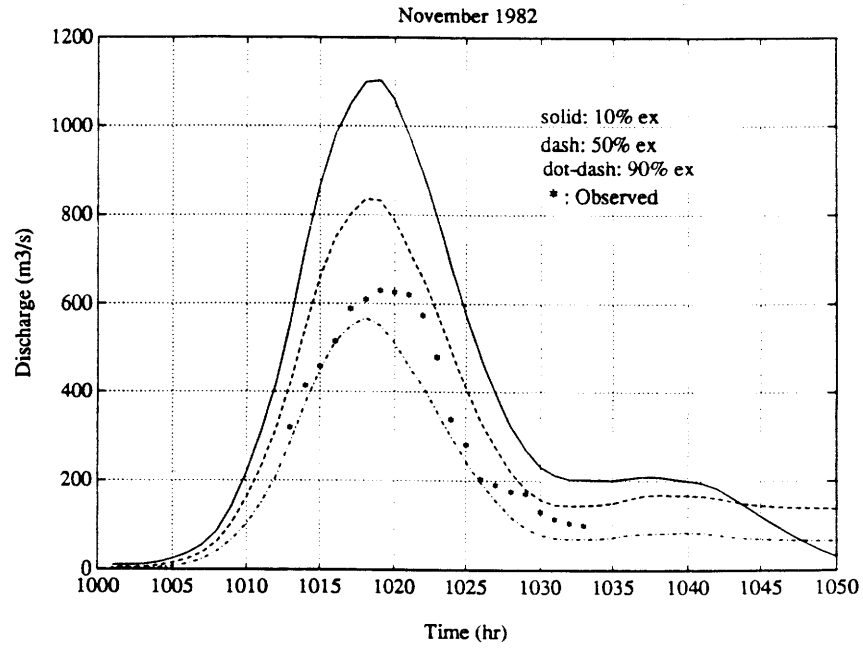


***Figure 3.37**: Observed and simulated hydrographs for the storm of November 1982. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

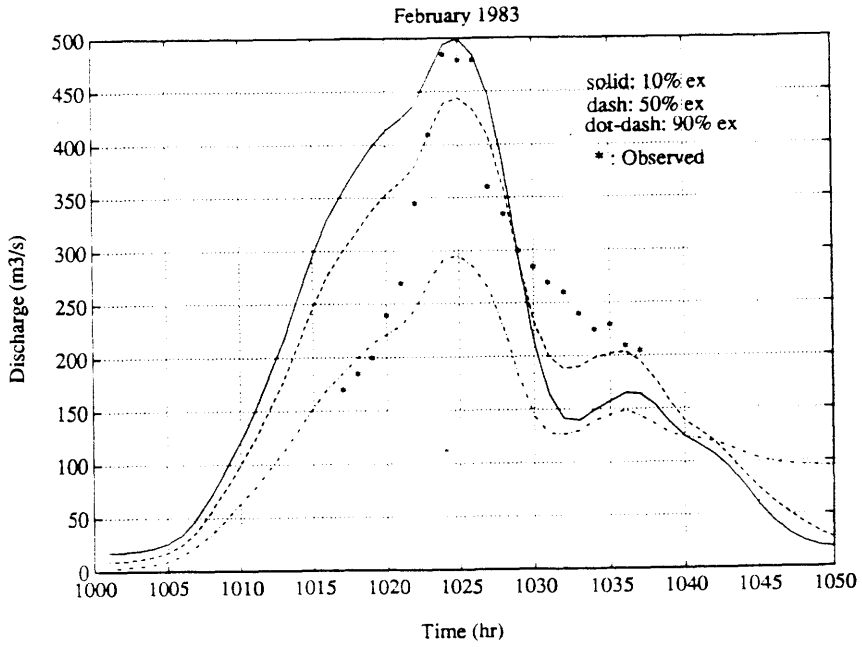


***Figure 3.38**: Observed and simulated hydrographs for the storm of February 1983. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

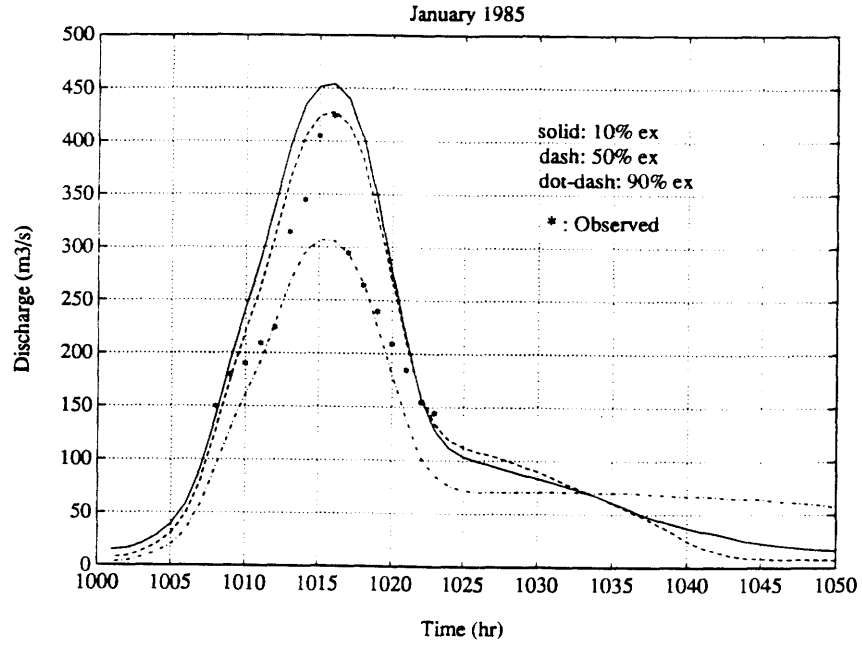


***Figure 3.39**: Observed and simulated hydrographs for the storm of January 1985. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

A subjective evaluation of the results of the calibration is presented in Table 3.9. Two qualitative variables are included in the analysis: peak and shape. Peak includes the ability to reproduce the values and specially the timing of the observed peak discharge. Timing is considered more important than absolute value, because the simulations cover a wide range of possible values, due to the uncertainty of the initial state. Shape is an overall evaluation of how the simulated hydrographs reproduce the total runoff volume and its distribution in time. The table also includes an estimation of the initial state that leads to the best fit between observed and simulated hydrographs.

Table 3.9 *Calibration. Summary of results*

Storm	Peak	Shape	Initial state	Result
February 1977	Fair	Fair	50%-10%	Fair
January 1979	Good	Good	50%	Good
November 1982	Fair	Good	90%	Good
February 1983	Good	Very poor	50% ?	Poor
January 1985	Good	Fair	90%-50%	Fair

Peak value and time to peak are reasonably well reproduced, but the overall shape of the simulated hydrographs differs considerably from that of the observed ones. Total runoff volumes obtained in the simulations appear to be larger than observed ones, but no conclusive assessment can be made due to the lack of continuous streamflow recording. In two cases, January 1979 and February 1983, the simulated hydrographs rise before the observed ones, suggesting a drier initial state and higher initial abstractions in the basin. Observed hydrographs also show an earlier falling limb, but this aspect is less clear.

The adjustment of the baseflow recession cannot be assessed conclusively, because streamflow measurements are discontinued very early. It seems that the early stages are acceptably well reproduced by the model, and the fact that the best adjustment is obtained with a very high anisotropy ratio is remarkable, hinting for a strong lateral flow. However, these simulations also illustrate a weak aspect of model behavior. The distributed model is event-based, and it only deals with storm water, whereas in reality baseflow recession is the combined consequence of storage of 'old' and 'new' water in the basin. As shown by the figures, simulated baseflow recession in the drier basin state usually lasts longer, since, in the model representation, the basin has a higher capacity to store 'new' storm water. The wet initial state does not store so much water, and, although baseflow rates are higher, the model, which only uses storm water, cannot maintain baseflow for a long time. This result of the model is in contradiction with experience, since it is clear that, in the same basin, drier basin states originate shorter baseflow recessions.

3.2.3 Results of the evaluation step

Five storms were originally reserved for the evaluation step. The evaluation storms are listed in Table 3.10. The model was run with the optimum parameter set presented in the previous section, and results were compared with observed discharge data for the five storms. No previous attempts were made to run the model for the evaluation storms with any parameter set.

Table 3.10 Storms in the evaluation set

Storm	Total rainfall depth (mm)
December 1968	31.77
January 1969	41.56
December 1975	51.67
December 1976	26.61
November 1987	103.03

The results are presented in Figures 3.40 to 3.44. Table 3.11 presents a subjective evaluation of the results, considering the same criteria used in the calibration step. The first general comment is that, although the differences between observed and simulated hydrographs are apparent,

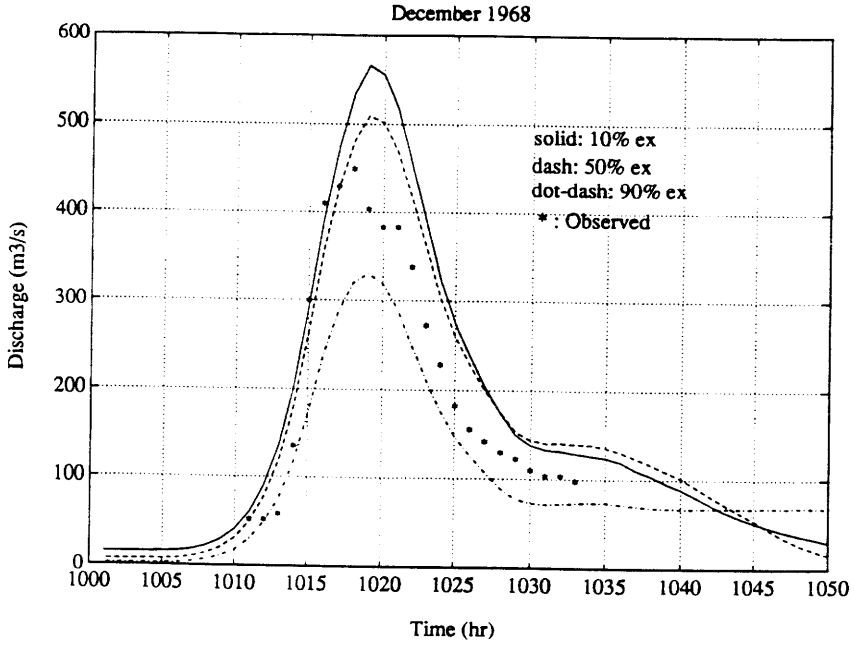


***Figure 3.40**: Observed and simulated hydrographs for the storm of December 1968. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

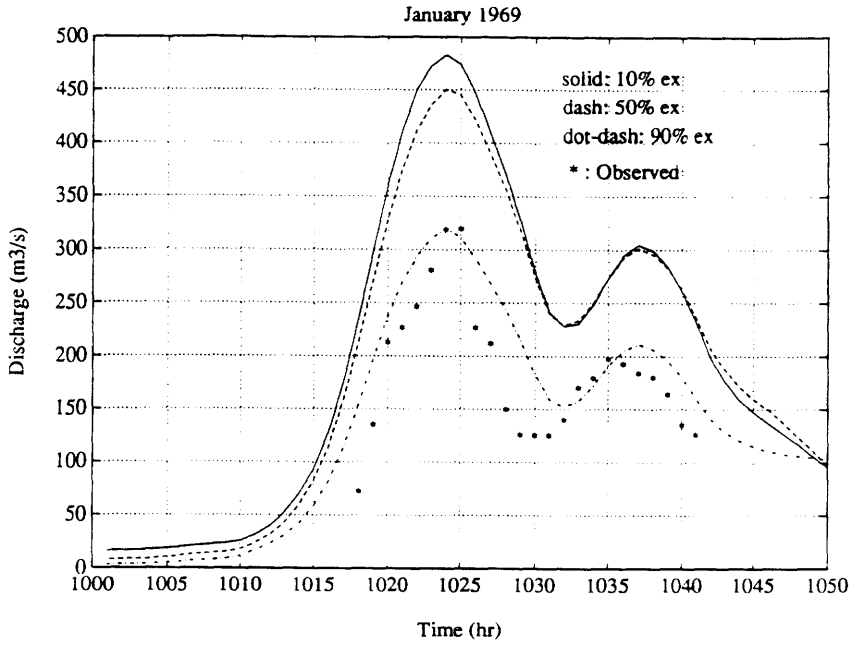


***Figure 3.41**: Observed and simulated hydrographs for the storm of January 1969. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

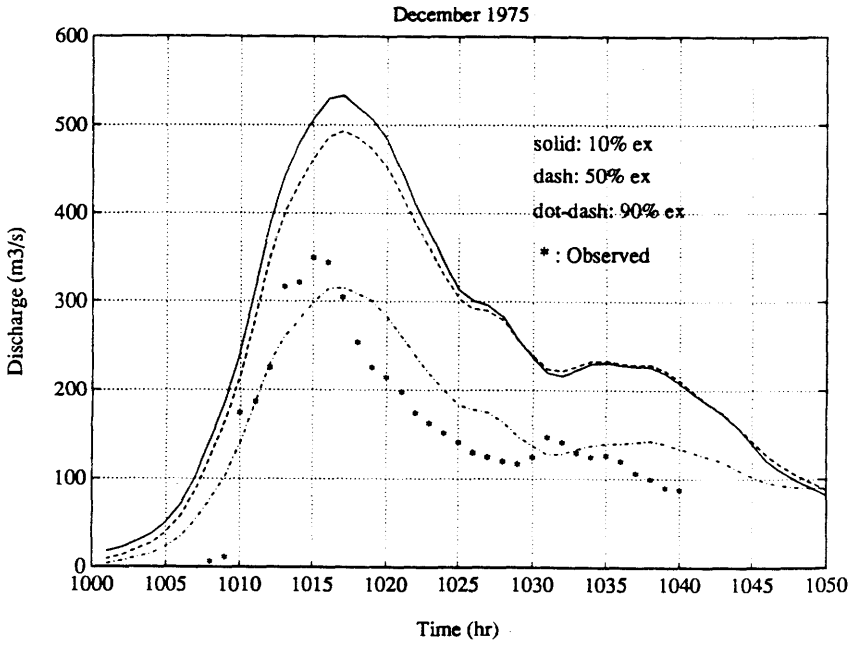


***Figure 3.42**: Observed and simulated hydrographs for the storm of December 1975. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

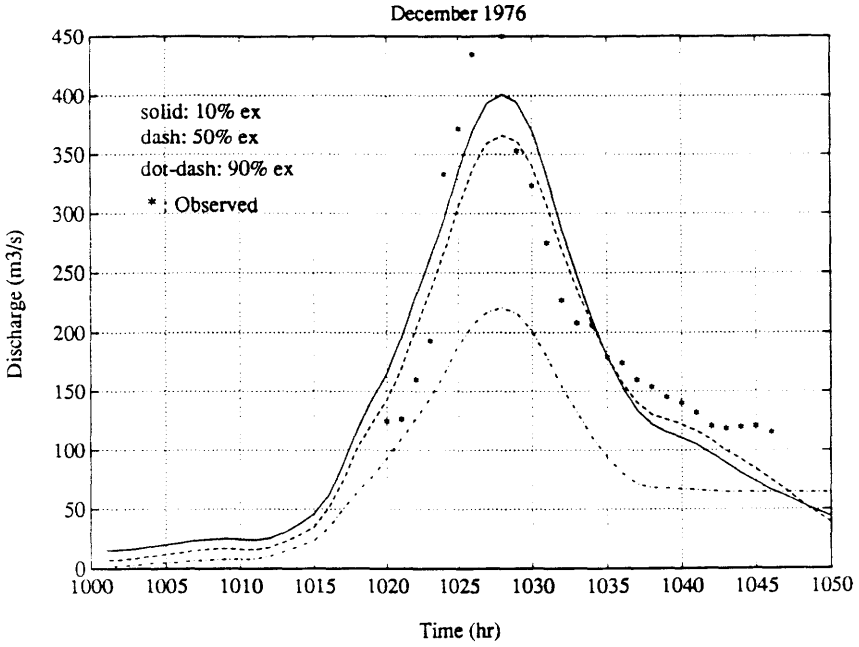


***Figure 3.43**: Observed and simulated hydrographs for the storm of December 1976. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

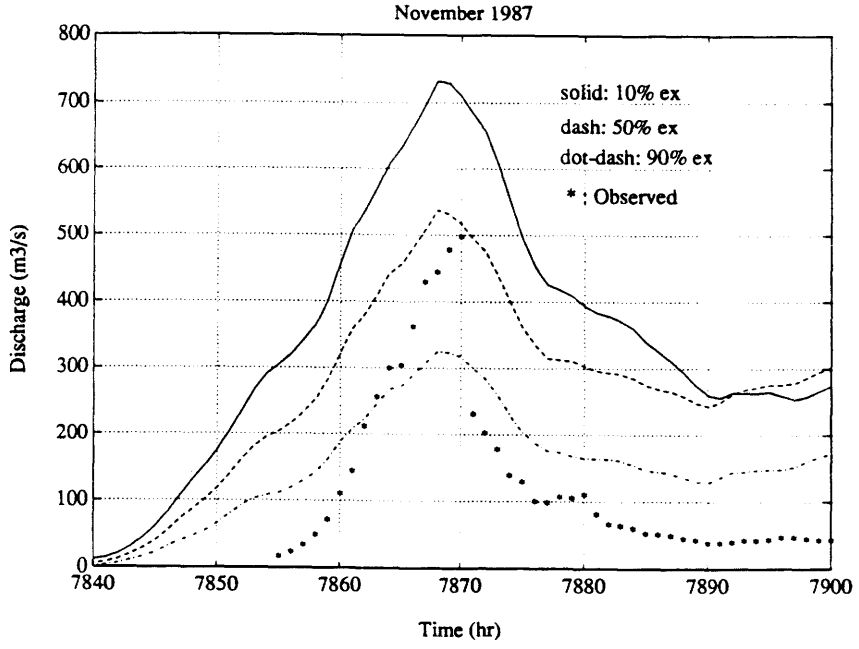


***Figure 3.44**: Observed and simulated hydrographs for the storm of November 1987. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

model performance with the evaluation set is not considerably worse than with the calibration set, which suggests that the calibration process was successful in the sense of estimating the best parameter set for the basin. If these other five storms were included in a new calibration process, the values of the parameters probably would not change very much.

Table 3.11 Evaluation. Summary of results

Storm	Peak	Shape	Initial state	Result
December 1968	Poor	Fair	90%-50%	Fair
January 1969	Good	Good	90%	Good
December 1975	Poor	Fair	90%	Poor
December 1976	Fair	Fair	<10%	Fair
November 1987	Fair	Very poor	>90%?	Poor

Compared to the calibration set, a greater dispersion with respect to the initial state is observed in this second set. While the December 1976 storm appears to correspond to a very wet initial state, the storms of January 1969 and December 1975 suggest a very dry initial condition. In general, model performance is acceptable in all storms except in November 1987, in which the observed hydrograph clearly crosses the simulated hydrographs for two initial states. The beginning and the end of the hydrograph correspond to an initial state drier than that of 90% probability of exceedance, whereas peak flow corresponds to the average initial state. The model clearly does not capture the dynamics of the basin in this case.

After the evaluation step was finished, data were received corresponding to a new storm in the Sieve basin. The storm took place in November, 1991, and the quality of the data was significantly better than that of the previously available storms. Rainfall information was still

obtained from a raingauge network, but network density was considerably higher. A total of 30 raingauges were available, and 6 of them were located inside the basin. Total rainfall depth in the basin was 111.14 mm. Streamflow data were also of higher quality, since continuous streamflow recording was available.

The model was run for this recent storm, and the results are shown on Figure 3.45. The results are very encouraging. The model reproduces the shape of the hydrograph with reasonable accuracy, and model performance is better than in the other cases of the evaluation set and even the calibration set. The simulation suggests a relatively dry initial condition, which is consistent with the time of the year (beginning of the rainy season). The timing of the rising and the falling limbs of the hydrograph is acceptable, and the adjustments of the interstorm periods

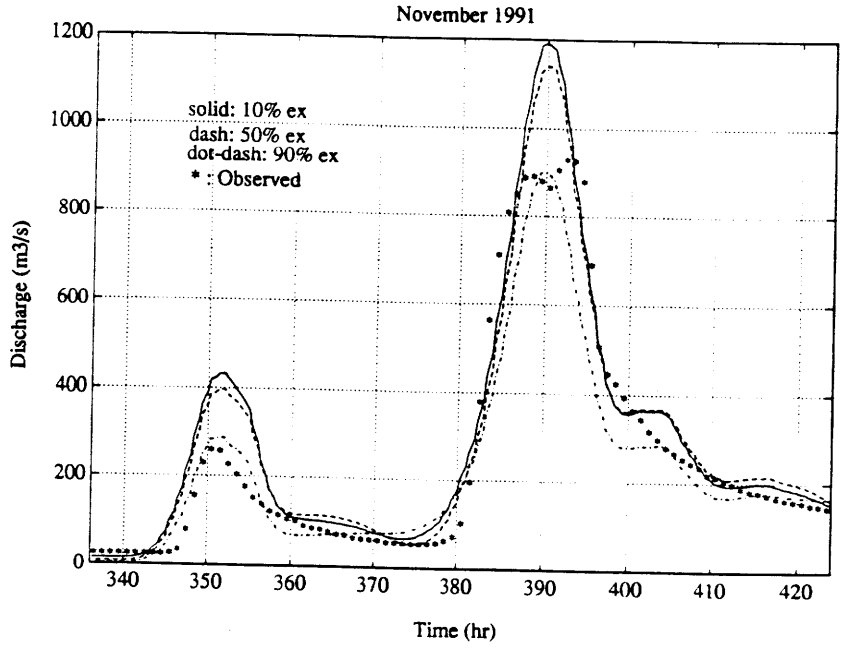


***Figure 3.45**: Observed and simulated hydrographs for the storm of November 1991. Simulated results correspond to initial states with 10%, 50% and 90% probability of exceedance.*

are excellent, specially if we consider that baseflow recession is generated through physically-based mechanisms using the same parameters as other model processes, and not through conceptual recession equations with specific parameters to fit baseflow.

Although this good model performance with a good data set could very well be just a serendipitous coincidence, it strongly suggests that model performance can be considerably improved with better data. In particular, distributed rainfall information obtained from radar pictures is crucial to evaluate the response of the different areas of the basin to rainfall irregularly distributed in space.

CHAPTER 4

Real-Time Use of the Model

In this chapter the real time operation of DBS is presented. Modeling capabilities offered by the basin simulator are used to build an interactive real-time environment for flood monitoring and forecasting. The package is called Real-time Interactive Basin Simulator (RIBS). Requirements for real time operation of a distributed, physically-based model are analyzed first, and a description of the solutions adopted is presented next.

4.1 Real-time operation of physically-based models during floods

This section discusses the problem of real-time flood forecasting in a generic sense. Since real-time flood forecasting is a very broad term, which may mean very different things in different contexts, the first subsection is dedicated to present the scope of the type of problems addressed in this work. In the second subsection we review the requirements of a flood forecasting system, which are taken as generic design goals for the RIBS environment.

4.1.1 Scope of the problem

In a broad sense, the goal of flood forecasting systems is to capture all available information during a flood situation and process it to different levels of complexity in order to present the user with information that can be included more directly in the decision making process. Two issues are of interest: how real-time information is captured and how it is processed to facilitate decision making. The information available is usually rainfall, either from a raingauge network or a meteorological radar, and streamflow data at different points of the basin. Since the relationship between rainfall and streamflow is complex, decision making is very difficult without the use of modeling tools to translate rainfall information into streamflow forecasts. Once streamflow forecasts at points of interest are available, they can be related to possible flooding problems. A real-time flood forecasting system must combine a real-time data acquisition system and state-of-the-art hydrologic modeling to provide the decision maker with the best information possible.

This work deals with the modeling aspect of the flood forecasting problem. More specifically, it is focused on the use of physically-based computer models in decision support systems for flood forecasting. In physically-based modeling, an attempt is made at understanding and reproducing the physical processes that take place in a river basin as it responds to intense rainfall. Since the behavior of the basin is extremely complex, strong assumptions and great simplifications are required to formulate the problem in terms that can be represented and simulated in a computer. Insufficient knowledge of the physical processes, assumptions and lack of adequate calibration data introduce errors in the

modeling process which lead to uncertainties about model results. However uncertain, model results are still usable by the decision maker because they enclose our best knowledge to interpret the available data. Since information systems normally include real streamflow measurements, model results can be compared with observed data, and hydrologic judgement can be applied to extract valuable information from the model behavior.

Since the decision-making process must be carried out in real time, an adequate software environment is very important to facilitate the task of model users, specially if the operation involves the use of complex and data-intensive distributed models. The objective of this work is presenting a methodology to develop software for physically-based models that are to work in this context. The final goal is not only to implement the model described in Chapter 2, but also to create a flexible and versatile software environment in which the model can be used under a wide variety of conditions and in which the user can have access to all the details of the modeling process in real time. This work provides fully-computerized support for all aspects of the decision-making process during a flood, and therefore the software presented here is designed to interact not only with human end-users but with other software modules which may be included in a larger decision-support system.

With respect to the physical system, the temporal and spatial scales of the problem should also be delimited. The structure of the forecasting system is strongly conditioned by the time of response of the basin. The river basin acts as a reservoir, delaying and dampening the effect of rainfall as runoff is transported through the drainage network. That effect is crucial in flood forecasting, since it enables the modeler to

produce forecasts of future outputs based on past inputs on the basin. The relative importance of future inputs on future outputs is marked by the time of response of the basin. For small basins, future outputs are almost exclusively a function of future inputs, and therefore the flood forecasting problem is in practice a rainfall forecasting problem. For larger basins past inputs have greater importance on future outputs, and reasonably good estimates of future streamflow can be produced based on past rainfalls, provided that a good rainfall-runoff model is available. For even larger basins, rainfall-runoff modeling becomes less relevant, since upstream discharge data are usually available. For these very large basins, the flood forecasting problem is largely a flood routing problem. Here we concentrate on midsize basins, with a time of response between 6 and 24 hours, for which the analysis is largely based on rainfall-runoff modeling, and our problem is to design and build a real-time system in which the rainfall-runoff model described in Chapters 2 and 3 is used at its maximum potential.

4.1.2 Requirements of a real-time flood-forecasting system

The best way of stating the design objectives for the RIBS system is to list the requirements that a successful real-time flood-forecasting system should meet. Some requirements, such as model updating, are classical in the field, and have been a main concern of research in real-time hydrology for a long time. Other features, such as output versatility, have been traditionally considered as less important because their value is usually associated with off-line modeling. However, recent progress in computer hardware and software suggests that it may soon be possible to

include our current off-line calibration and simulation methodologies in real-time systems. There are also requirements, such as real-time data acquisition, which are not hydrologic in nature. They respond to practical needs of real-time computer systems, but they have a significant impact on the global performance of the decision-making process.

Next, we present a summary of the basic features that a real-time software package for flood monitoring and forecasting should have. Although the implementation of RIBS presented in this work does not address all these requirements specifically, the importance of all of them has been considered in the design process, and special care has been taken to make specific modeling needs compatible with more general operational requirements.

On-line operation

The first requirement is that the system must operate on line with a real-time data-acquisition network. The system is intended to process data at the rate at which they are collected by the network. Channels of communication and operational methodologies must be established to process data in real time without the direct intervention of a user to manipulate the information. Since network reliability is not perfect, the system should provide for mechanisms to check data accuracy and fill in the blanks when necessary. It should also be ready to operate flexibly under variable conditions, since regular data arrival to the processing center cannot be guaranteed. In addition to on-line capabilities, the software package should also provide calibration tools, enabling off-line

simulations with historic or synthetic data sets for calibration and training purposes.

Real-time response

The system should be able to produce results in time to be useful. The time available for operation of the system is variable, depending on basin size and time of response. Other variables are also important, such as the time necessary for civil defense actions: warning dissemination, evacuation, emergency rescue services, etc. In general, there is a lower limit to the anticipation with which a system can operate, marked by the inherent time of response of the decision-making process. That limit may be set between one and two hours. Streamflow forecasts of less than two hours of lead time are usually of little use to the decision maker, because by the time he or she has finally access to them a comparable lapse of time has passed since the original data were obtained. By real time operation we mean the capacity to produce streamflow forecasts of at least two hours of lead time is less than one hour of processing time.

Discrimination between measured and forecasted rainfall

From the standpoint of rainfall-runoff modeling, measured and forecasted rainfall are treated in the same way, but from the standpoint of decision making, the value of both analyses is significantly different. Although measured rainfall is not free from uncertainties, it provides more reliable information than forecasted rainfall. The nature of rainfall forecasts is inherently more speculative and uncertain than the

estimation of past rainfall based on measurements. As a consequence, the results obtained using forecasted rainfall are always less reliable than those obtained using measured rainfall.

It is therefore important that a real-time system can discriminate between the fraction of basin response that is due to measured rainfall and the fraction that is due to forecasted rainfall. Both pieces of information play a different role in the decision making-process, and the system should be able to separate them explicitly. The system should also offer the user a variety of possibilities to play with measured and forecasted rainfall independently and to explore the implications of uncertain rainfall forecasts.

Real-time updating

The goal of real-time flood forecasting systems is to provide information to make decisions during situations of crisis. Given the current state of the art in hydrologic modeling, it is unlikely that a perfect real-time forecasting system can be developed. Imperfections of current modeling methodologies, lack of sufficiently long historical data and limitations in measuring technologies are the main causes that lead to less than optimal performance of real-time systems. Fortunately, real-time data collection networks also offer measurements that can be used to test model results. Any operational scheme to use a model in real time should also provide means for updating certain aspects of model behavior according to its performance. The system should account explicitly for the possibility that successive model predictions do not agree with observations. A higher-level process should be in charge of monitoring

model results and comparing them to observations, deciding whether an improvement in model performance can be obtained and how.

State information

The role of models in decision-support systems is strongly conditioned by how good model performance is. In fields where well-tested models are completely reliable, such as spacecraft guidance systems, for instance, the model can be viewed by the decision maker as a black box, since the decision can be reliably based on model results. In other fields, such as operational hydrology, the state of the art of modeling techniques cannot guarantee that models are always accurate, and the interpretation of model results is a very important part of the decision-making process. Of course, the fact that modeling is not without errors does not mean that models cannot be used as decision-support tools. Models encompass the best available knowledge to deal with the evolution of physical systems, and, to the extent that they capture the relevant features of the behavior of physical systems, they are extremely helpful tools in the analysis of complex situations. But, in order to attain that goal, hydrologic models must diversify their outputs, and offer information about basin state in addition to the traditional streamflow forecasts.

Distributed models may be the answer to the problem. They attempt to capture the relevant physical processes that play a role in basin response, and they actually build a computer model of the basin that can be explored in different ways. Their output is not only a hydrograph at the outlet of the basin; it is composed of a number of variables that describe basin state and characterize its response to future rainfall. If model

performance is good enough to capture the main features of the processes that lead to local saturation in certain areas of the basin and is successful in describing the dynamics of the runoff generation mechanisms, the knowledge of basin state as described by the model can be of valuable importance to a decision maker.

Explanation facilities

In order to be adequately interpreted, models must explain their results. They must provide the users with tools that enable them to understand how the models got their results and why. Under some circumstances, models produce inaccurate results because they are working under unexpected conditions. Given the adequate tools, experienced users can detect that circumstance and still extract useful information from model results. In order to be useful, a real-time system based on the use of a distributed model must provide easy access to the basin representation used by the model and allow the user to follow model inference in time. The access to the information must be versatile, flexible and selective. A user should be able to specify how he or she wants to consult model results and basin state from a wide array of possibilities. The user should be able to decide which variables are of interest to him or her at a given moment and be able to change his or her strategy to access model output at any time during the storm according to variable needs.

4.2 System concept

The integrated software package for real-time flood forecasting is conceived as a high-level manipulation of basic modeling capabilities considering operational hydrologic needs. This work is simultaneously concerned with two methodological aspects of the problem of designing an efficient flood forecasting system: the hydrologic approach and the software engineering approach. We try to couple them together, so that the hydrologic analysis of the problem is mapped onto a correlative software organization. The general idea is to combine the important contributions in the fields of theoretical computer science and numerical hydrologic modeling, to mitigate the problems traditionally found when only one of the approaches is applied in isolation.

Recent real-time packages for physically-based simulation (Widman et al., 1989) are built around abstract software notions, with most emphasis being placed on symbol manipulation. These developments have produced advanced intelligent packages with remarkable flexibility. However, the attempt to isolate the software design from the specific domain in which it is intended to operate may lead to unnecessarily complex systems, in which the potentials of physically-based simulation are constrained by the needs of abstract representation. Moreover, the prototype architectures proposed in the literature do not contemplate data intensive applications, and their solutions are usually organized around in-memory data storage and manipulation, which limits their scope to small applications.

Physically-based simulation models, on the other hand, no not take advantage of recent advances in software development techniques, and

remain constrained by implicit limitations imposed by software design and implementation in FORTRAN language. These models are successful in the sense that they perform their computations efficiently and accurately, but their interaction with the user is limited to the initial data and final results, with no other possibility for user access to internal states than the cumbersome trace functions, only valid for debugging purposes. Software is organized upon the concept of function or subroutine, which strongly limits the possibilities for reusability and expansion.

4.2.1 Design considerations

We present a model development methodology in which software design is effectively coupled to physically-based modeling with the final objective of a better user access to internal model representations, both for calibration and real-time use purposes. The methodology also addresses the problems of data-intensive applications and coordination of several processes that must be running simultaneously.

The final goal of the design is to specify a computer environment in which the distributed rainfall-runoff model described in Chapter 2 can be used to assist in decision making during a flood situation in real time. We first review the two main elements that must be considered in the design: data and procedures. Since distributed models deal with large amounts of data, their effectiveness requires that the data be adequately structured and organized. The use of distributed models involves the interaction with a hydrologic data base, which plays a central role in the design process. The system must deal efficiently with storage of and access to different

kinds of data which may evolve in time. Data complexity also requires an efficient interaction with the user, who must be aided in the task of understanding and evaluating large quantities of data.

The second element of importance are the computer procedures that generate and operate on the data. In order to simplify the design, the global problem must be decomposed into simpler tasks, implemented as computer procedures. The procedures must be run simultaneously and must operate concurrently on the data base. The need for problem decomposition is not so much a consequence of the size of the global task as it is a consequence of its complexity. In order to attain flexibility, the system must react to the current situation, rather than follow a pre-specified sequence of operations. Higher-level control modules must be able to handle other lower-level modules (which implement specific operations) in a symbolic manner.

According to these two basic elements, system design is based on the concepts of procedural and data abstraction (Abelson and Sussman, 1985). Different tasks must be performed during model operation. The tasks are treated symbolically as abstract procedures, regardless of the particular implementation that is chosen for a given task. Similarly, data are grouped in conceptual units that are handled symbolically, regardless of the particular format or storage environment selected for them. The key idea of model design is to identify self-contained data sets and procedures that can be manipulated by a high-level manager that makes decisions. This conceptualization allows for an easy global design that deals with tasks and data at a high level, without having to consider implementation details for them.

4.2.2 System architecture

This section presents the philosophy of the system architecture adopted for RIBS. System architecture follows a hierarchical design, as shown in Figure 4.1. The foundation of the system is composed of three layers: modules, functional units and software objects. All layers share the same conceptual design, but correspond to different levels of analysis, characterized by the detail of problem decomposition. The modules match mostly the end-user view of the system, the functional units correspond to the hydrologist's analysis and the software objects are the programmer's interpretation. RIBS is not a single program, but a family of programs. A final version of the RIBS system is composed of several modules operating on a shared database and controlled by a general manager.

A module is an operational unit on its own that accomplishes one of the basic tasks introduced in the previous section. Modules are self-contained software units that interact with the whole system by operating on the common database and by sending messages to the general manager or to other modules. Modules can and should operate concurrently, advancing in time as new data become available. In the RIBS environment there are rainfall acquisition modules, rainfall forecasting modules, rainfall-runoff transformation modules and user interface modules. It is conceivable that several modules be defined to accomplish the same task, either to attain redundancy in the process or to offer different analyses of the same problem.

The building blocks of the modules are functional units, that are combined to perform the task. Functional units represent hydrologic concepts that are directly mapped into software functions. They

correspond to real actions with hydrological meaning operating on the data structures. Functional units can be symbolic or numeric in nature, depending on whether they perform a qualitative or a quantitative analysis. The main difference between modules and functional units is

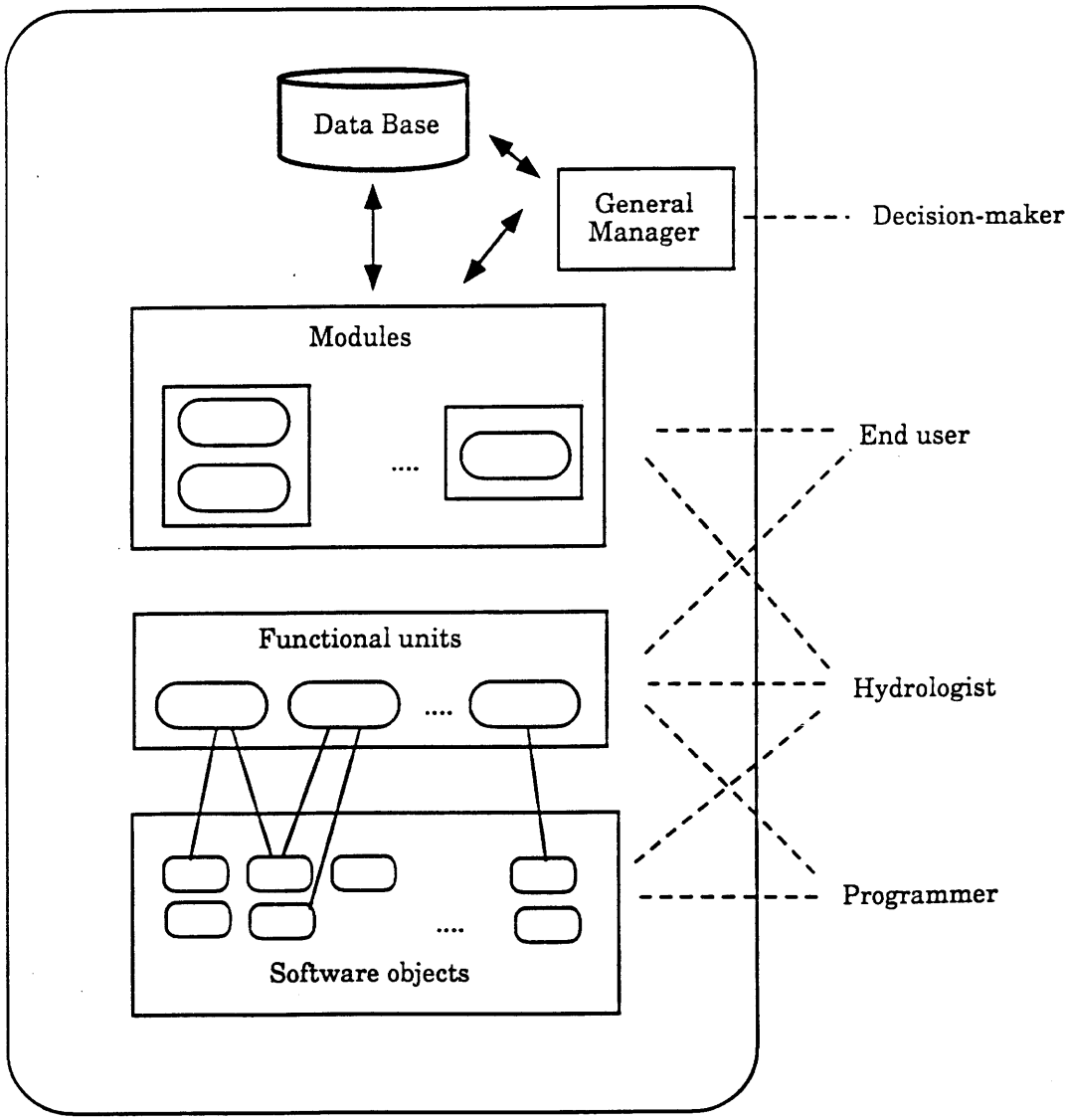


***Figure 4.1**: Schematic representation of the architecture of RIBS*

that functional units are interrelated, whereas modules are entirely self contained. Therefore, functional units are building blocks that do not produce final results by themselves, but contribute to the final outcome. Functional units must be organized within a module to cooperate in the generation of results. They are usually tightly coupled, in the sense that there are strong interactions among them and intensive data transfer or sharing.

On the lowest design layer, the environment is supported by a number of software objects with behavior associated to them, following the object-oriented design methodology. Objects are shared by different functional units, so all modules are built upon a common structure and share the same understanding of the data involved. In addition to facilitating the design process and allowing reusability, object-oriented development is a crucial factor in the construction of an interactive model interface, since the same processing units are used in the modeling modules and in the user interface modules, and the user is given access exactly to the same operations on the data as the model is.

In this section we provide a general description of the basic modules considered in RIBS and a discussion of the high-level management of individual modules. The hydrologic analysis presented in the previous section suggests four groups of procedural modules and a number of static and time dependent data structures. We focus on what the modules are expected to do and how the manager controls their behavior, regardless of how those goals can be achieved. Since a full implementation of all features of RIBS is beyond the scope of this work, the software development presented here focuses only on two of the four modules that integrate RIBS, namely the rainfall-runoff transformation

module and the user interface module. However, a basic implementation of the other two modules and of the general manager is also provided for completeness. A description of software components and internal organization is given in Chapters 5 and 6. There are also a user manual and a design document, included as appendices, where low-level details are presented.

4.2.3 Process handling

The term "procedural abstraction" refers to how a complex problem is decomposed into individual tasks and how the system handles these tasks symbolically to accomplish its goal. Following the paradigm of procedural abstraction, the design of RIBS is modular, with high-level modules represented by executable programs and low-level functional units represented by object methods. We concentrate here on the manipulation of modules by the system, and Section 4.4 discusses the functional units that integrate the two main modules.

Functionally, modules can be viewed as black boxes. They retrieve information from the database, operate on it and store results on the data base again. Modules are integrated in a higher system architecture, and hence they cooperate to attain global goals. Their operation must therefore be concurrent. In order to allow for concurrency, once a module has finished its task it must notify to the manager or to the other modules affected that the state of the database has changed. Proper channels of communication must be open to grant the possibility of module interaction. Internally, modules are composed of elements tightly coupled together, but the group of modules is a loosely coupled collection of

elements, in the sense that each module is self-contained and module interface is kept as minimum and as standard as possible.

Three main tasks are necessary in a real-time flood forecasting system: data acquisition, rainfall forecasting and rainfall-runoff modeling. In addition to that, the user must also gain access to the internal results through an adequate user interface. Therefore, four basic types of processes are considered in the design, as represented in Figure 4.2.

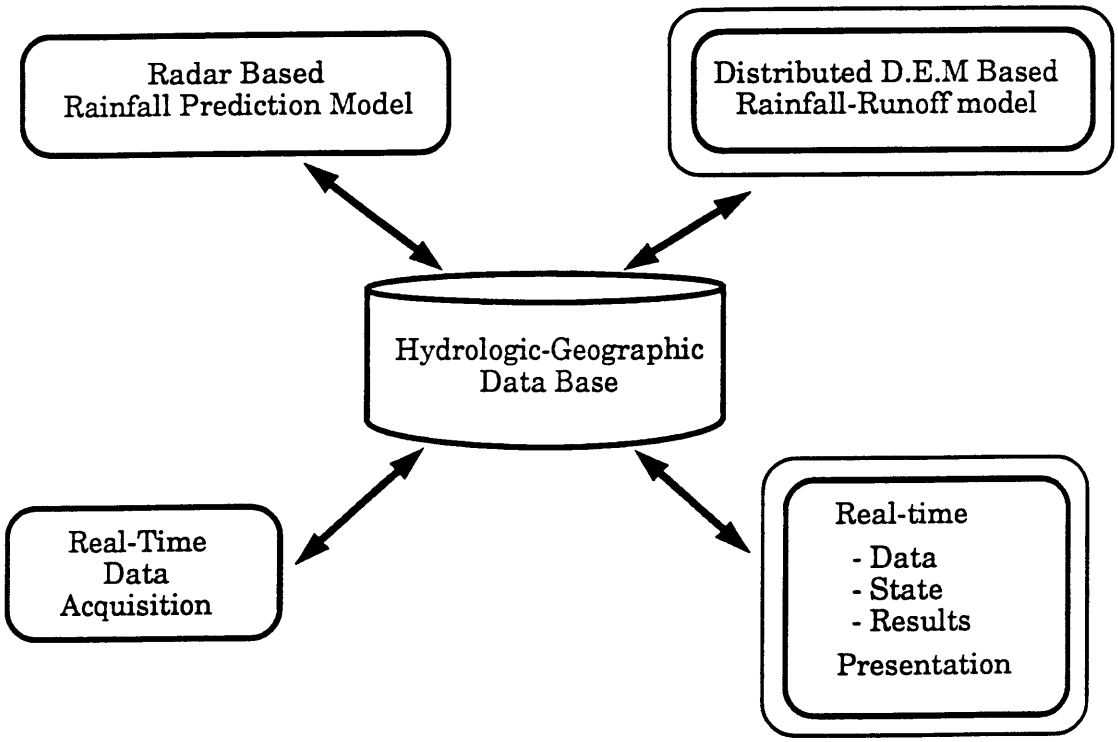


***Figure 4.2**: Processes in a real-time flood-forecasting system*

Rainfall acquisition modules

Rainfall acquisition modules (RAM) scan the real-time data acquisition network and extract rainfall information. The goal is to generate periodically a distributed description of measured rainfall. Rainfall information may come from a raingauge network, radar pictures or both. The RAM analyzes the raw information and makes it available to the rest of the system in a format which can be understood by all other modules. Each measured rainfall generated by RAM has a time tag, which is the time at which it was generated, and is assumed to correspond to the time interval between its time tag and that corresponding to the previous one. The RAM also takes care of data verification, checking input information for consistency and filling in the errors or blanks in the data received.

Different implementations can be considered for the RAM, depending on the data acquisition network available. The procedure or procedures selected should include redundancy whenever possible, since that reduces the probability of data blanks and offers better ground for consistency checks. The acquisition of information from the network is clearly a quantitative problem, in which the right algorithm has to be applied to interpolate values of the rainfall field between raingauges or to relate radar reflectivity at every point with rainfall intensity on the ground. The analysis for data consistency is a more complex problem, probably involving the comparison of measurements from several sensors of different type. Since it is in general an ill-defined problem, qualitative methods could be included to improve the performance of automatic quantitative procedures.

Rainfall forecasting modules

The rainfall forecasting modules (RFM) analyze measured rainfall and generate rainfall forecasts. They take as input the rainfall description generated by the RAM and, possibly, other meteorological information. The output is expected future rainfall, with some additional attributes, such as time of validity or probability of occurrence. The RFM may include one or several forecasting procedures, and it does not have to be limited to generating just one set of results. In fact, real-time, user-defined future rainfall alternatives may be included through an adequate interface module.

Rainfall-runoff transformation modules

The rainfall-runoff transformation modules (RRTM) transform rainfall information into streamflow predictions at one or several points within the river basin. A numerical model of the river basin is built to reproduce how the basin responds to rainfall. The most important characteristic that RRTM's must have is flexibility. RRTM's are used as a decision making tools. Their function is to translate rainfall information into predictions of streamflow at certain locations, and therefore their operation is conditioned by specific circumstances. Depending on the circumstances, the focus of attention will usually concentrate on different areas during a storm. If the situation is critical on an area, abundant and detailed information is required about a relatively reduced location, while the situation on the rest of the basin is comparatively less important. An

ideal RRTM must therefore have flexibility to offer streamflow at different points in the basin at the user's request, reacting to the evolution of the storm.

User interface modules

While the other three groups of modules are running, guided by the availability of new rainfall information, the user must also have access to intermediate calculations of the modules, and be able to obtain reports about different variables concerning basin state. The user interface modules (UIM) query the database to extract high-level descriptions of basin state, lumped and distributed attributes, etc. The interface modules must contemplate two types of users: software clients and human users. Software clients are other programs that use the functionalities provided by the UIM to access some aspects of model results. Human users are also interested in model results as intermediate or final output of the other software modules. The user interface should be designed to meet requirements posed by both groups of users. An adequate user interface design can greatly enhance the possibilities of communication between different software applications and facilitate the processes of model development and calibration.

There are two types of user interface modules: model driven and user driven. The model-driven interface is a unidirectional channel of communication between the model and the user. The model-driven interface presents result updates as the model progresses. All setups in the model-driven interface are defined before the beginning of the simulation, and model results trigger output presentations. Object-

oriented programming and multiuser operating systems offer the possibility of user-driven interfaces in which the user requests which results he or she wants to consult at any time. Setups in user-driven interfaces can be redefined during the simulation and the events triggering output presentations are external to the model itself.

4.2.4 Data handling

The second aspect that a large application has to deal with is data. This section about data handling discusses how different modules share a common language for data description through the process of data abstraction. At a high level, the application data are lumped into meaningful sets which are handled symbolically by the modules, following the concepts of object-oriented design. Since all modules are intended to share and transfer information, a common data storage and a universal data description are needed. However, every module may keep a particular internal data representation. That can achieved in practice through a series of libraries used to interact with the data structure adopted for external storage.

Data abstraction refers to the grouping of individual pieces of data into homogeneous units of distinct meaning (Gorlen et al., 1990). That collection of information configures an object which is manipulated as a whole by programs dealing with it. From the standpoint of physical storage, data structures are internal and external. Internal objects are data stored in memory, and correspond to software entities which also have a behavior associated to them. Internal data structures are the result of the development of object-oriented approaches within the

discipline of programming languages. The role of internal data structures in RIBS is described in Chapters 5 and 6, where software design is presented. In this section we concentrate on external data structures, which are data kept in external storage. Data models for external storage have followed a development parallel to that of programming languages, evolving from first-generation file systems to fourth-generation relational databases. The next generation of database technology will probably follow an object-oriented model (Kim, 1990), where in-memory software objects can be stored in an external database.

Practical reasons support the adoption of external data storage. Ideally, objects should be kept in memory to minimize input/output operations and reduce processing time. However, the amount of information handled by the system is clearly larger than the memory that can be expected in a workstation environment, and therefore, a solution involving external data storage is certainly justified. Data sharing is another reasons. Although there are available schemes to share data in memory, external data storage is a simple way of sharing data among a number of independent processes, as long as concurrent access to the files is prevented.

Data can be classified according to their formal structure or to their semantic content. The formal structure refers to the internal organization of the data, which in our case maps the structure of physical variables (distributed/punctual and static/dynamic data). The semantic content refers to the specific meaning of the information contained in the data structure (rainfall, discharge, elevation, etc.). According to the formal structure, two main types of data format are of interest in this application: the raster format, used to describe two-dimensional data structures and

the hydrograph format, used to describe time series of discharge and rainfall. These two and other auxiliary data formats are described in the Appendix 3. We concentrate here on the discussion of data types according to their semantic content, which is more relevant to the overall design of the application. According to the semantic content, the following data groups can be defined, as shown in Figure 4.3:

Basin description

This group corresponds to the static information required to describe the basin from a hydrologic point of view. It includes the geometrical description: location, shape, topography, slopes and other magnitudes derived from a DEM. Data about hydrologic attributes and properties, such as hydraulic conductivity, vegetation or soil texture also belong to this group.

Two types of data structures are used to describe the river basin: lumped and distributed. Lumped information refers to basin attributes which are independent from spatial location or, at least, uniform throughout the basin. One singe value is enough to characterize the whole basin. Distributed information refers to attributes that have spatial meaning and vary with location inside the basin. More than one value is necessary to specify distributed information. The storage requirements associated with a particular distributed variable depend on the format selected for data representation and in the nature of the data themselves. Distributed data may be originally distributed in nature, or may vary through an indirect dependence on a common index, such in the case of soil properties, which are a surrogate of soil types.

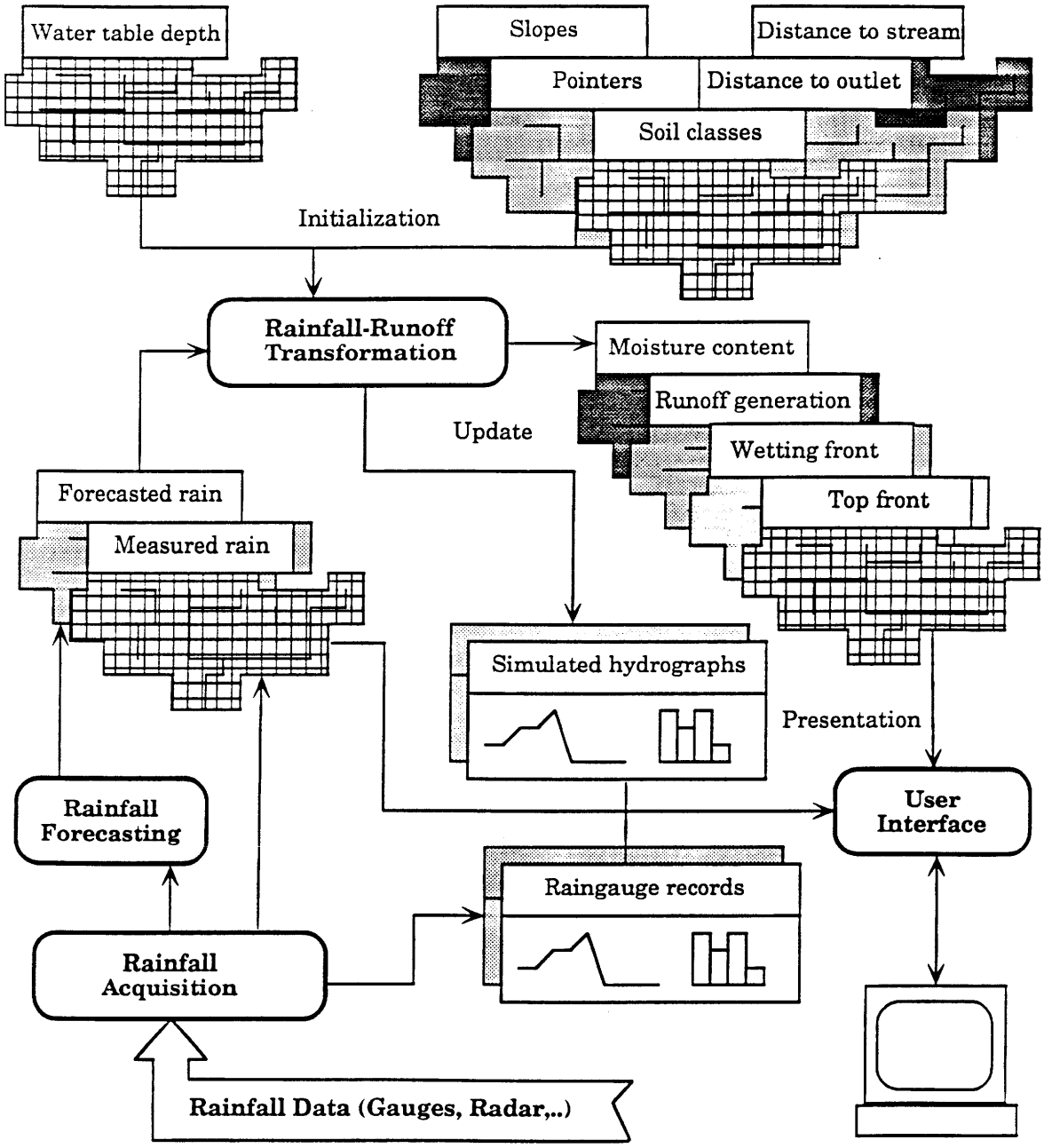


***Figure 4.3**: Data flow in a real-time flood-forecasting system*

The software package presented here is built around the use of a particular distributed model, and therefore it was considered that data description should accommodate to the basin representation adopted by the model. Therefore, the spatial structure that supports distributed data representation is a rectangular grid. Every node in the grid is assigned a value, following the raster format for data description proposed in the literature about spatial information systems. The raster representation is adequate, since most hydrological properties vary continuously throughout the basin. A vector format does not offer a significant improvement from the standpoint of storage requirements, and introduces additional complications from the standpoint of computational efficiency.

Measured and forecasted rainfall

Rainfall information should be stored in a format that can be directly processed by the model. Rainfall information is distributed in nature. However, some measurement schemes are based on the collection of punctual information in a number of locations, and the application of interpolation or other numerical procedures to obtain a continuous spatial distribution. In other schemes, distributed information is directly gathered in the form of digital maps from radar or satellite sensors, or a mixed solution is adopted. In any case, maximum overall efficiency is obtained if details about the measurement scheme are encoded only in the rainfall acquisition module and hidden from the rest of the application. Therefore, the format for permanent storage of rainfall information should be independent from the original measurement format.

Observed and computed hydrographs

Hydrographs are an important piece of information in the system. Observed hydrographs are used by the model updating modules to estimate basin state and correct model performance. Observed and computed hydrographs must also be available for the end user. An interesting aspect that differentiates observed and computed hydrographs is that observed hydrographs have only one temporal dimension, which corresponds to the time evolution of the observed variable, whereas computed hydrographs are defined in two temporal dimensions: the time dimension of the discharge and the time at which the hydrograph was computed. Both temporal dimensions must taken into account in the data storage system.

Basin state

The term "basin state" refers to the information needed at any time to obtain the future basin response. The special formulation of the kinematic model of infiltration in terms of state variables allows the computation of basin evolution in terms of a previous state plus an incremental basin response. Knowledge of the state variables plus the rainfall and the runoff generation also permit to reproduce any derived concept defined in the model. In order to characterize basin state, two types of information are needed

a) State variables of the infiltration model in all the nodes of the grid. The state variables are necessary to compute the future evolution of the fronts and obtain the runoff generation capabilities at any point.

- Position of the wetting front in the basin $N_f(x,y)$
- Position of the top front in the basin $N_t(x,y)$
- Moisture content in the basin $M_t(x,y)$

b) Incremental hydrograph due to antecedent rainfall. That is the expected contribution to the global hydrograph as a consequence of the runoff generated by rainfall in the previous hours.

- Distributed runoff generation $R_f(x,y)$
- Previous incremental basin response $Q_p(t)$

4.2.5 General management

Modules perform meaningful operations, but they are embedded in a larger system which has more general goals than the individual goals of each module. In addition to the proper functioning of every module, it is the coordination of the individual modules what eventually produces a helpful decision-support system. The general manager coordinates the different modules and makes them compatible. The role of the general manager is important to optimize model performance according to the situation. The manager can schedule the tasks that it considers pertinent given the time available for operation and the current storm situation. Task scheduling through a manager also facilitates the possibility of responding to user's request about the global functioning of the system, since the role of the manager can be easily transferred to the user through an adequate interface.

The architecture of the system is built around the common data base shared by all modules. The operation of the modules on the database resembles that of the blackboard architectures proposed in the literature of knowledge-based systems (Nii, 1986). A central structure (blackboard) supports the application data, and a group of knowledge sources can operate on the data on the blackboard to generate more elaborate versions of the original data. Each module is specialized on a certain type of operation, which contributes to some aspect of the final goal. Modules take data from the blackboard, operate on them and place their results back on the blackboard again when they are finished.

The sequence of operations to follow cannot be specified a priori, because it usually depends on the contents of the blackboard, but some guidelines are known about which modules should be triggered whenever an event occurs in the blackboard. The general manager contains the knowledge and methodology to operate with the modules according to the contents of the database in order to satisfy the general goals. Three aspects of the operation of the general manager are of interest: module coordination, time control and inter-module communication.

Module coordination

Two basic options are available to control module coordination: centralized management and hierarchical management. In a centralized-management architecture, all modules are triggered by a process, the central manager. Modules also communicate the completion of their tasks to the manager process. The manager in turn knows which other modules should be notified of the event, and sends messages to

them. For instance, whenever the rainfall acquisition module generates new rainfall information, it sends a message to the manager. The manager knows that active processes using the measured rainfall are the rainfall forecasting module, the rainfall-runoff transformation module and a graphic display of the user interface. It sends messages to those processes to notify them that new information is available. The recipient processes, which were in a dormant state waiting for that information, get reactivated when they receive the message from the manager. They then read the information from the database and operate on it, each one notifying the manager when finished.

In a hierarchical-management architecture modules control one another. Instead of a unique parent process and a group of child processes, a tree of interrelated processes is formed, in which any process can trigger child processes and keep control of them. The individual modules have knowledge about which modules are affected by their behavior, and propagate event messages accordingly. In our previous example, the rainfall acquisition module would be the parent of the three processes which depend on its actions, and it would send messages directly to them whenever new rainfall information is available. The child processes can in turn create other processes and control them through direct messages.

Both architectures have advantages and disadvantages. The centralized architecture provides a better control of process evolution, but it requires a more efficient communication among the processes. Hierarchical management is simpler, but it does not allow for closed loops in process dependencies. In particular, it is difficult to include user decisions about process evolution in real-time. The use of one or other type

of management depends on the complexity of the problem and on the nature of the relationship between modules. For simple problems in which the sequence of operations can be specified a priori, hierarchical architecture is an efficient and straightforward management method if there are no mutual dependencies between processes. More complex problems, involving decision making about task pertinence and priority would probably require a centralized architecture, including knowledge bases to reason about process control.

Time organization

The control of time is an important aspect of the general management. In addition to triggering or reactivating the adequate processes as a reaction to the occurrence of events, the manager must also keep track of time evolution. Since the system is advancing in real time, the time reference is dynamically changing as the storm progresses. In order to allow for process interaction, a unified time reference must be shared by all modules. Moreover, the system must also be able to backtrack in time and offer images of past situations and forecasts to compare with the actual storm evolution. If unacceptable model performance is observed, the manager may decide to change some model parameters, return to some previous state and continue model evolution from there. Time organization should not be contemplated only as a centralized control of linear time evolution, but as a branched process in which model evolution can be interrupted at some point and reassumed from a previous time with different model settings.

Inter process communications

Monitoring the time evolution of a flood is essentially a cyclic task, in which several processes repeat the same activities as new data become available or as new forecasts are required. Best use of computer resources is achieved if those processes are run in parallel in a multiuser environment. Management of parallelism requires that the processes establish channels of communication between them. The UNIX programming environment offers several possibilities for inter process communication, which essentially represent the three basic modes for interprocess communication: pipes, message queues and shared memory.

The simplest mode of process communication is the use of pipes, in which a parent process sends output to a child process, which accepts it as input. Communication can only be established between two processes in an unidirectional way. A message queue is a communication channel that connects several process in a multidirectional way. All connected processes can put messages in the queue and can retrieve messages from it. A coding scheme enables processes to read only messages of interest to them. A message queue can therefore support any type of inter process communication, provided that the size of the messages passed between processes is small. A third method of communication between processes is a shared memory area, which is generally used when the size of the information shared is large and access time is important. For the scope of RIBS, a design based on either pipes or messages queues can adequately support the required communication between modules, since messages are usually short. A shared memory area would only be required for

implementation reasons, for instance if different modules share large amounts of information and the memory available in the system is limiting. Pipe communication is adequate in a hierarchical management strategy, where processes only receive information from their parent process. The centralized management strategy requires a message queue, since bidirectional communication with the manager is necessary.

4.3 RIBS prototype

This section describes a prototype implementation of RIBS built around the distributed basin simulator described in Chapter 2. We present a real-time operational loop that uses modeling capabilities of the DBS to assist in real-time decision making. The goal is to use the distributed model to monitor basin evolution during a storm and provide detailed descriptions of basin state at the user's request. Only part of the requirements for a real-time flood-forecasting system described in Section 4.1 are satisfied in this particular implementation. The application is not complete, since no real-time update is contemplated, and only a very simple coordination module is defined, but it illustrates basic capabilities and shows how different software modules are shared by the rainfall-runoff model, the user interface and the general manager. Model structure and operation are designed so that the system can be easily expanded to address other goals involving rainfall forecasting and model updating. Modeling objectives are defined first, and then the architecture design is presented. A description of the multiple modes of use of the

system is given in section 4.4.3, and as a first step towards code design, the functional units involved are described in Section 4.4.

4.3.1 Modeling objectives

The objective of this prototype implementation of RIBS is to configure a system that can operate the distributed model in real time, addressing the issues of time coordination and easy user access to model results. The work is focused on two areas: real-time hydrologic modeling and user interface. The treatment of the other two areas, real-time rainfall acquisition and rainfall forecasting is merely symbolic, but they are included in the final package for the sake of completeness. From the point of view of hydrologic modeling, DBS is used to fulfill two basic goals: simulation with observed rainfall and simulation with forecasted rainfall. Both objectives are interrelated, but they are different and involve distinct operating procedures. The objective from the standpoint of model-user interaction is to provide a versatile user-driven interface with enough explanation facilities to understand model evolution in real time.

Simulation with observed rainfall provides an image of basin state up to the current time and a forecast of the minimum streamflow that can be expected. Given that model structure is incremental, the simulation with observed rainfall should advance in parallel with the storm, obtaining a new incremental basin response every time new rainfall observations are available. Simulation with forecasted rainfall provides images of future basin states and forecasts of streamflows for future rainfall. The user should be able to compare the results using observed and forecasted rainfall, to obtain an idea of how future rainfall might affect sreamflow. It

is also useful to include in the comparison the prediction made in the last time step, in order to appreciate the dynamics of storm evolution. The specification for the operational use of DBS are therefore simple: DBS should advance in parallel with the storm. Whenever new rainfall arrives, it should compute basin evolution corresponding to that rainfall and store results. Then, it should compute basin evolution with the rainfall forecast and present all results to the user. At the beginning of the next time step, the state of the basin at the end of the simulation with observed rainfall should be recovered to continue the evolution with new rainfall.

The user interface should have two versions, simultaneously available to the user: a model-driven interface and a user-driven interface. The model-driven interface must offer results as they become available from the model. Results include at least hydrographs at different points in the basin (defined by the user) and the state variables of the model: position of the fronts and moisture content. Information about measured and forecasted rainfall and runoff generation is also of interest. Since that information is too abundant, the user should also be able to select which of those variables are displayed at any given time, and configure the user interface screen at his or her will.

Another version of the user interface is also necessary. In addition to being interested in the latest update of basin state, the user might also want to consult basin state at a previous time, or time evolution of any model variable. In order to get that, a user-driven interface is also required. This interface would respond to user requests to display basin state or any derived variable at any time up to the present, generate hydrographs at the user's request or display the time evolution of any

variable. Both interfaces should be easily operated through mouse-driven menus, avoiding cumbersome text-based interaction or complex command syntax.

To complete the system, a simple mechanisms to emulate real-time arrival of rainfall information should also be made available. The rainfall generation program must also include a simple module to generate rainfall forecasts.

4.3.2 Architecture

This section describes the architecture adopted for the prototype implementation of RIBS. The implementation follows the general guidelines described in section 4.2. A central hydrologic database is shared by a number of modules that operate on it. The overall design is conceived with a workstation environment running the UNIX operating system as target platform. Although individual modules could also run on an advanced personal computer, the UNIX operating system is best suited for its facilities for multitasking and interprocess communication. Three aspects of model architecture are of interests: database configuration, module definition and coordination between modules.

Database configuration

The term 'database' refers to all data handled by the application. Since the volume of data involved is very large, in-memory data storage is not feasible, and therefore an external data storage scheme is needed. The solution adopted for external data storage is based on UNIX files. Other

solutions, such as a database management system or a geographic information system are also equally feasible, but file interaction provides the best efficiency in terms of storage requirements and access time. Traditional advantages of database environments, such as consistency checks, concurrency control or indirect queries are not relevant in this particular application. Most of the interaction involves reading or writing fixed portions of well-identified information with a stable structure, and therefore lumping data in files is a solution adequately suited to the task. Data are also well structured and permanent in nature, and their organization in a hierarchy of directories is perfectly feasible. Moreover, the use of operating system files offers the best solution in terms of portability, since no additional software is required to handle data storage.

The solutions adopted for database organization, file naming conventions and file format are detailed in Appendix 3. Files are of two types: static and dynamic. Static files correspond to permanent concepts, while dynamic files correspond to variables that evolve in time. Different files are used to store variable values at every time. According to the structure, two basic file formats are defined: raster format for two-dimensional variables and time series format for hydrographs. Additional formats for trace files are also defined, although they are not included in the general database. Except for the time tag in dynamic files, file names are defined by the user. The user is also free to structure files within the directory tree according to his or her own interpretation of the semantics of the data. Graphic tools are provided to display files of different formats. All formats but the raster are text-based, and can be directly edited by the user. A special compressed format was adopted for

raster files, and convenient tools are provided to create and edit files of this type.

Module definition

A total of seven executable modules conform the RIBS application, organized as shown in Figure 4.4. Three groups correspond to the hydrologic modeling area and four groups correspond to the user interface area. The core of the hydrologic modeling area is the program *dbsim*. *dbsim* is a distributed basin simulator that implements the model described in chapter two. It is built to interact with the database, reading rainfall description files and generating files describing basin state variables and output hydrographs. Auxiliary programs are *rain_gen_gauge* and *pixel*. *rain_gen_gauge* is a program that emulates the arrival of rainfall information in real time. It reads hyetographs from several raingauges and generates distributed rainfall maps over the basin using a simple interpolation algorithm. *rain_gen_gauge* also generates rainfall forecast, based on a simple *AR(1)* process. The program *pixel* implements the one-dimensional model of infiltration for a soil column. It was developed mostly for calibration purposes, in order to have a tool to analyze the isolated behavior of one singe pixel of the basin.

The user interface area is composed of four modules. The core program is *xbview*, an interactive X Windows-based application which provides access to basin state at any given time. Auxiliary modules are *xrasgraf*, *xhydgraf* and *xpixgraf*, interactive applications that present raster files, hydrographs and pixel state to the user. User interface applications appear on the screen as independent windows composed of a

drawing area and a menu bar. The windows can be manipulated individually by the user, and the user can dynamically reconfigure the working screen according to his or her changing needs. Graphic displays have zooming capabilities and most user interaction is mouse-driven.

Process coordination

There is a high-level process coordination, which combines the individual capabilities of the different modules and coordinates them all towards a concrete goal. Since the proposed model operation (simulation with measured rainfall - writing of results - simulation with forecasted rainfall) is very simple, a hierarchical management strategy was adopted, and no specific manager module was programmed. However, the design of individual modules is such that a more elaborate operational strategy is feasible and relatively straightforward to implement.

Each module corresponds to a UNIX process. Processes are organized in a hierarchical structure, which represents the logical dependence between hydrologic processes. Figure 4.4 shows the hierarchical dependency for the real-time mode of operation. The parent process is the rainfall generator module, (*rain_gen_gauge*) which is therefore the top-level manager. The rainfall generator controls the rainfall-runoff module (*dbsim*) and several interface modules to display measured and forecasted rainfall and hyetographs at selected locations. The rainfall-runoff module controls interface modules to display results and a real-time basin access module (*xbview*). Apart from that, another basin viewer module can be started independently to access model results

at a prior time. The basin viewer module also controls another set of interface modules to display results.

The basic settings of the modules are specified through environmental variables. Since in UNIX children processes inherit their parents' environments, this solution guarantees that all processes in the same hierarchy share the same context, in terms of paths, directories, file

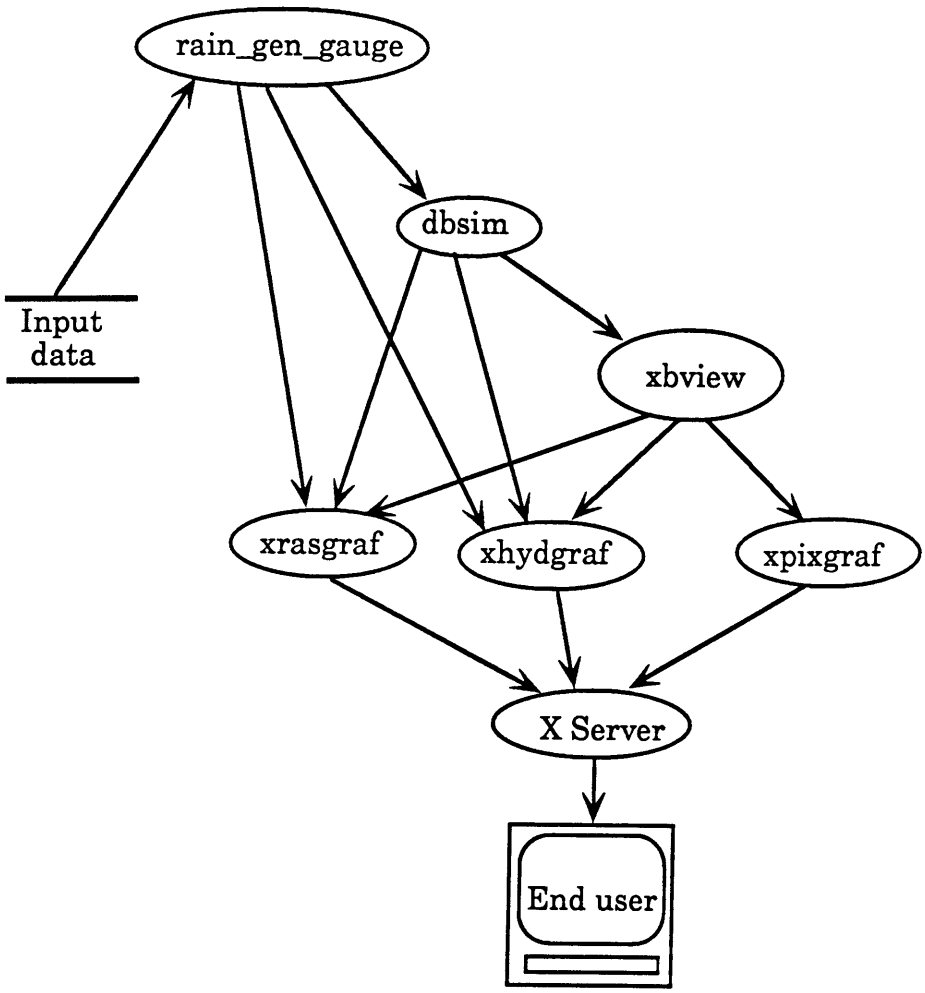


***Figure 4.4**: Process dependencies for the actual modules of RIBS*

names and model variables, and minimizes errors due to the use of modules in different contexts.

Communication and control between different modules is implemented through pipes. The parent process opens a pipe to the child process and writes information concerning the child process into the pipe whenever adequate. Once the child process is started, it waits for instructions from the parent process on the standard input. The rainfall-runoff module reads the time tag of new rainfall information, the display modules read the name of the file to be displayed, and the basin viewer reads the last time step finished by the process. The basic utilities provided by the *X Server* are used in the interface modules to accept simultaneous input from the screen and from the standard input, thus allowing for interactive operation controlled by both the user and the parent process.

4.3.3 Modes of operation

This section describes the user perspective of the prototype implementation of RIBS. Although the intended use of the package is real-time assistance in decision making during a flood, other user needs should also be taken into consideration in order to configure a useful application. We comment on the different functionalities offered by the package, and how the same code is adapted to meet different requirements. Four scenarios for the use of RIBS are contemplated: (1) *calibration mode*, in which the model is used to perform several simulations with the same storm data and different model parameters, (2) *simulation on line mode*, in which the model in simulation mode is

connected to the real-time data acquisition module to present results in real time, (3) *forecasting on line mode*, in which the model also obtains and presents results using forecasted rainfall, and (4) *forecasting off line mode*, in which the model is run in forecasting mode, but with a pre-existing rainfall information. The user manual included in Appendix 2 gives the specific operations required to run the model in each of these modes. Here we discuss the user needs that each module is intended to fulfill and the rationale behind the design.

Calibration mode

The calibration mode is the use of the distributed model in the most simple form. The user is interested in generating effective runs of the model with different parameters, but no monitoring of partial results is required. The input data have been previously generated, and only a limited number of results are of interest. Ease of operation and minimum execution time are the relevant requirements for the calibration mode.

There are two main options to design an interface for model calibration: interactive mode or batch mode. In an interactive calibration, the user is presented with one or several screens in which he or she can change model parameters according to the results. In a batch mode, the user sets a script file with a number of prespecified runs, and checks the results after the process is complete. Interactive calibration is adequate for fast processes, since the user is usually waiting for results in front of the screen. Batch calibration is intended for more complex processes, for which the waiting time for the process to complete makes if unfeasible for an interactive mode.

In the case of RIBS, the response time of the model for a complete storm is so long that the user cannot wait in front of the screen for model results. Running the model on the background is preferable, and batch operation was selected for the calibration mode. The goal is to allow the user to make several simulations without having to interact directly with the computer. With this operation scheme, the process can run on the background of a multiuser machine without direct user supervision. Another concern of the calibration mode is that a single run of the model generates a number of files which require a sizeable amount of disk space. If the user is only interested in final results, and no detailed analysis of intermediate states is required, the abundance of output files can be a great inconvenience and file writing operations deteriorate the time efficiency of the model. Therefore, the calibration mode also includes an optional inhibition of periodic writing of model results.

To facilitate the interaction between the computer and the user, the different modules of the program are set to take arguments from the command line, following the classic UNIX style. The user can prepare a shell script for each model run with the adequate settings for the environmental variables and the appropriate instructions on the command line of *dbsim*. The user can optionally select the display of model results on the screen, but model advance in this mode is usually so fast that the time between successive updates is not enough for a detailed analysis of model results. User access to model results is provided through the user-driven off-line interface, the basin viewer.

Simulation on-line mode

The simulation on-line mode consists on running the model with data acquired in real time. Only measured rainfall is used, and the user interface offers expected hydrographs corresponding to measured rainfall, and a detailed description of basin state. The simulation on-line mode is adequate when no reliable rainfall forecasting procedure is available. The rainfall generation module and the rainfall-runoff module should be activated. Two types of interfaces are provided for this case. While RIBS is running, the ability to monitor partial model results is provided by the model-driven interface, which presents results as they become available during the simulation. Channels of communication are available to present rainfall, basin state variables, runoff generation and hydrographs at prespecified locations. The user can select which information should appear on the screen as the model advances by activating the corresponding environmental variable. The user-driven interface can also be used by activating the environmental variable corresponding to the basin viewer.

Forecast on-line mode

The forecast on-line mode is the final intended mode of operation of RIBS. In this environment, the package is used to analyze input information, generate rainfall forecast and evaluate basin response to measured and forecasted rainfall. The results obtained from the model are twofold. First, the user obtains streamflow forecasts in all points of interest. Combining different future rainfall alternatives, associated to

different probability levels, several possible scenarios can be analyzed. Second, the model also provides information about basin state at current and future times. Information about basin state is also very meaningful to the decision maker, since it characterizes the runoff potential of the basin for future rainfall. The user interface is analogous to the previous case, and offers the possibility to explore basin state and obtain a detailed report of several derived variables.

Forecast off-line mode

The forecast off-line mode simulates a real-time forecasting session, but using previously obtained rainfall information. The goal is to provide model users with an instrument to analyze and evaluate model performance in real-time, either using previously recorded data sets or synthetic storms. The forecast off-line mode includes the rainfall-runoff module with the two user-interface modules. It is in practice like the on-line mode, but changing the input channel. Rainfall file names can be given through the standard input or read from the rainfall directory.

4.4 Functional units

We present in this section several examples of the functional units, building blocks of the RIBS modules. The functional units represent high-level computation blocks which are shared by different modules of the environment. Two types of functional units are described in this section: rainfall-runoff functional units and user interface functional units.

Rainfall-runoff functional units represent the behavior of physical systems included in the rainfall-runoff simulation model, while user interface functional units correspond to the graphic operations necessary to display a certain type of information on the screen.

4.4.1 Rainfall-runoff functional units

There is a one-to-one correspondence between rainfall-runoff functional units and the basic processes identified in the hydrologic analysis of the problem, and therefore there is a strong coupling of software design and and hydrologic modeling. Software objects represent hydrologic entities and functional units reproduce their behavior. This design facilitates the processes of hydrologic modeling and software development, since both can evolve in parallel, introducing incremental changes on previously developed concepts.

The rainfall-runoff functional units correspond to the individual blocks of the nested operational scheme. A structure composed of three nested loops is used, as shown schematically in Figure 4.5. The one-dimensional model of infiltration is the basic building block of the basin-scale runoff generation process, which, together with the routing scheme, represent the basic model capabilities presented in chapter two. These basic capabilities are used to evaluate basin state evolution and incremental basin response every time step. We call that process the "computational loop". On a larger time scale, that basic loop is repeated to evaluate basin response to a period of uniform rainfall, in what we designate the "rainfall loop". Another external loop (the "forecasting

loop") can also be defined to evaluate basin response to different alternatives of forecasted rainfall.

The one-dimensional model of infiltration

This functional unit coincides exactly with the behavior of a soil column such as that defined in Chapter 2. The objective is to integrate model equations to obtain the dynamics of front position and the evolution of the moisture content of the column. The process follows three steps. In the first step, the updated front positions are evaluated, using Equations (2.16) and (2.17). These equations are integrated using an explicit numerical scheme, where the values obtained in the previous step are

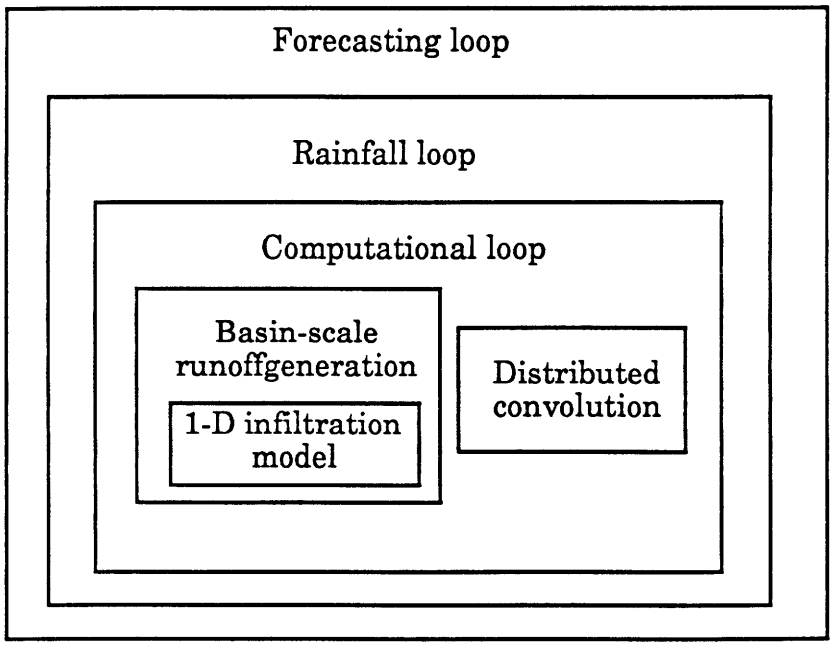


***Figure 4.5**: Organization of hydrologic functional units*

used to evaluate the new front positions. Given the new front positions, the infiltration capacity is evaluated according to Equations (2.28), (2.29) and (2.30). Moisture balance is then performed, comparing rainfall to infiltration capacity to obtain infiltration and infiltration excess runoff. An important aspect of the unit is to control numerical issues that may appear due to the unstable nature of model equations under certain conditions.

The basin-scale runoff generation

This functional unit represent the process of runoff generation at the basin scale. It involves the simulation of subsurface flow transfer among pixels and the use of the previous functional unit to evaluate local runoff generation at every grid point. Given the rainfall rate and the values of the state variables in the basin, the excess runoff is evaluated in three steps. In a first step, the disequilibrium moisture transfers Q_{pout} (x,y) are evaluated, using the expression (2.27) for every pair of contiguous pixels. That modifies the pixels' moisture content, but the value of their state variables is not changed. The second step computes the evolution of every grid point using the infiltration functional unit, and lateral flow using Equation (2.20). Infiltration, disequilibrium moisture transfers and upstream subsurface flow are moisture inputs to the soil columns and downstream subsurface flow is the output from subsurface flow. The net result is the new moisture content. If the new moisture content is greater than soil capacity above the wetting front, return flow is evaluated as the difference between moisture content and soil capacity, and moisture content is set to soil capacity. Total runoff is the sum of infiltration excess

runoff and return flow. The final output is the distribution of runoff in the basin $R_f(x,y)$.

The distributed convolution

The distributed convolution reproduces the process of water transport from the hillslope to the basin outlet. This unit takes the distribution of runoff generation as an input and applies the routing process described in Section 2.3 to obtain discharges at any point within the basin. The output is the incremental basin response corresponding to the time step, $q_{\Delta t}(t)$.

The computation loop

The computation loop refers to the operations necessary to obtain the evolution of the basin corresponding to a period Δt. The computation loop has two phases:

- Evaluation of runoff in the basin $R_{\Delta t}(x,y)$.
- Evaluation of the incremental response $q_{\Delta t}(t)$ for every location of interest.

The result of the computation loop is the incremental hydrograph in points of interest plus the evolution of basin state during the time step.

The rainfall loop

Rainfall loop is the processing of basin evolution and incremental response for a period of constant rainfall intensity. The rainfall loop is invoked every time new rainfall information becomes available. It is

assumed that the time of validity of rainfall T_R is considerably larger than the computation time step. Therefore, the core of the rainfall loop is the simulation of basin evolution repeating the computation loop. However, other elements are required as preprocessing or postprocessing operations of the rainfall loop.

The first operation required is the activation of the loop. Since the system is supposed to run in real time, a mechanism must be established to trigger the rainfall loop whenever new rainfall becomes available or a higher-level controller decides that the evaluation of a rainfall loop is pertinent. Once the process is started, the sequence of operations to follow is simple. First, information about the new rainfall must be obtained. The information required basically refers to time span of validity of the rainfall and details about how to read it (namely, file name if the implementation adopted is a file system). Once rainfall is read, the basin state corresponding to the time of beginning of the rainfall loop must be recovered from the data base. Timing compatibility has also to be considered, since only an integer number of cycles can be evaluated. This involves either the decomposition of the time of validity of rainfall in an integer number of computation time steps or the comparison of the time span of the rainfall with a simulation clock to obtain an acceptable approximation in terms of number of cycles. The computational loop is then invoked, and, once finished, basin state and incremental response may be stored for later use if needed.

The forecasting loop

The forecasting loop is the processing of basin evolution and incremental response for one or more alternatives of estimated future rainfall. It is an outer loop built over the rainfall loop. A rainfall forecast is composed of an ordered sequence of one or more rainfall values over time. A rainfall loop is defined for every rainfall value in time. However, two aspects are different. First, the forecasting loop considers rainfall in the future, and therefore the validity of that rainfall is known a priori. Secondly, it loops over different rainfall alternatives for different moments in time.

4.4.2 User interface functional units

Since the user interface is intended to support communication between the user and the rainfall-runoff model, it is built upon the hydrologic functional units. However, specific functionalities are needed to allow for interactive access to model results. The functional units of the user interface are therefore the graphic tools that present model results on the screen. In addition to low-level capabilities common to all interactive graphic systems, the user interface requires high-level tools to display persistent objects stored in the database and software objects maintained in memory by the system. The interface is built upon three basic types of display: pixel display, hydrograph display and basin display.

The pixel display

The pixel display presents a graphic display of the state of a soil column, according to the one-dimensional model of infiltration presented in Chapter 2. It is the graphical presentation of the *Pixel* software object, discussed in Section 5.2. The pixel display represents the variation of the moisture profile with depth, including initial moisture content and moisture infiltrated during the storm. The profile also presents the position of the fronts and the value of the saturated moisture content. An example of the pixel viewer is presented in Figure 4.6. It can also generate a cross-sectional view of the hillslope, representing streamlines of infiltrating flow and displaying the position of the wetting front, the top front and the water table.

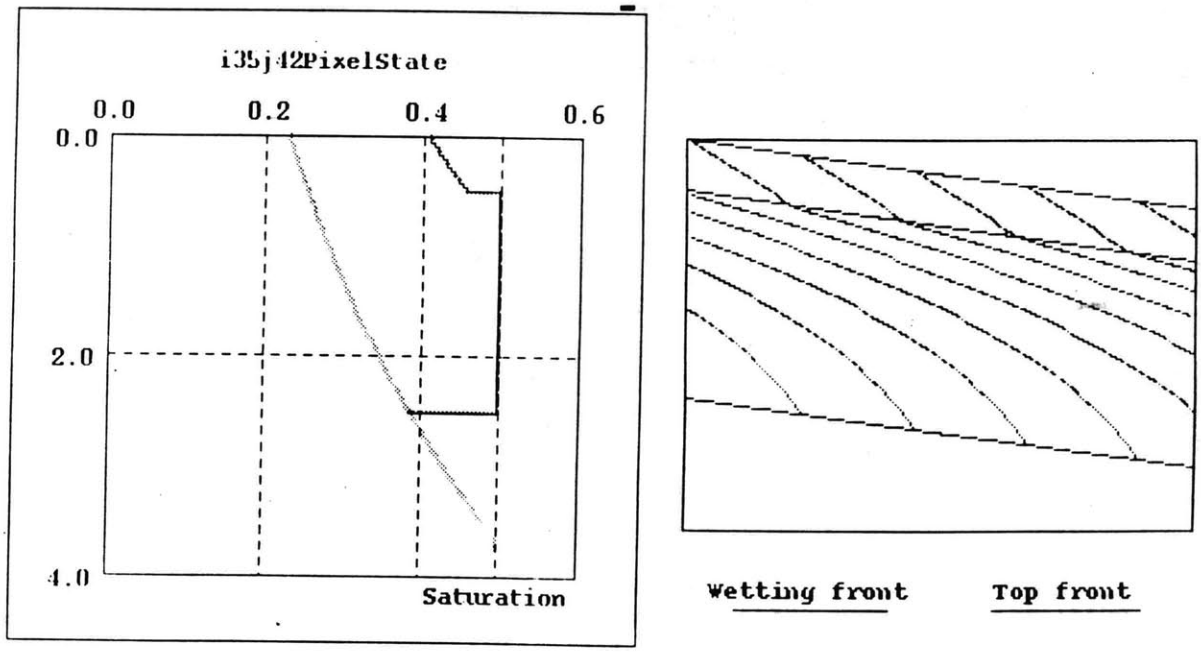


***Figure 4.6**: Example of a pixel display*

The hydrograph display

The hydrograph display is a graphic tool to display the time evolution of several hydrologic variables on the screen. In is the graphical presentation of the *hydrograph* objects stored in the database. Every variable represents a hydrograph at a point, and is composed of two values: rainfall in the contributing area and streamflow at that point. Streamflow is represented as a line on the lower plot, while rainfall is represented as a bar chart on the upper plot. Both time axes coincide. Each variable represented is assigned a color code. An example of the

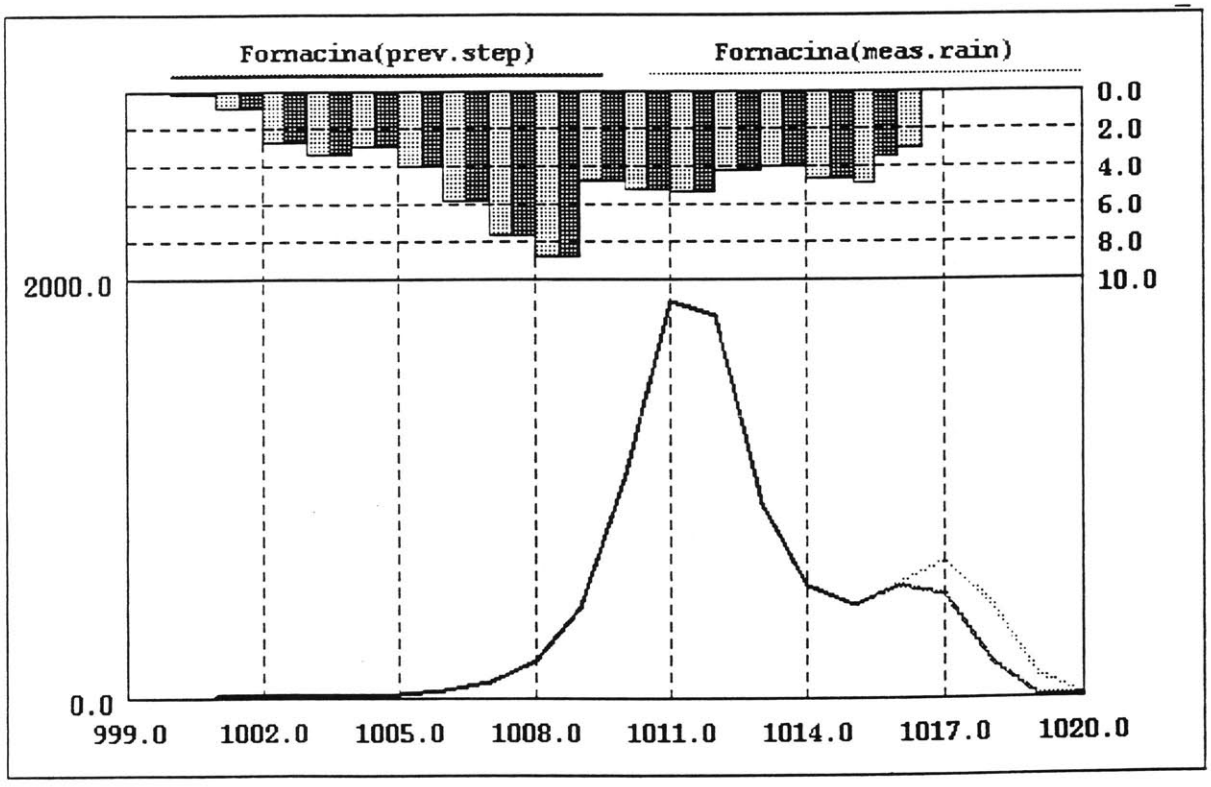


***Figure 4.7**: Example of a hydrograph display*

hydrograph viewer is presented in Figure 4.7.

The basin display

The basin display is a graphical display of the spatial distribution of a variable. It is the graphical presentation of the *raster* objects stored in the database. The values of the variable are represented as contour bands using a color code, following a user-defined color palette and scale. The color code is presented to the left of the graph, with a histogram of frequencies of occurrences of every color. Figure 4.8 Shows an example.

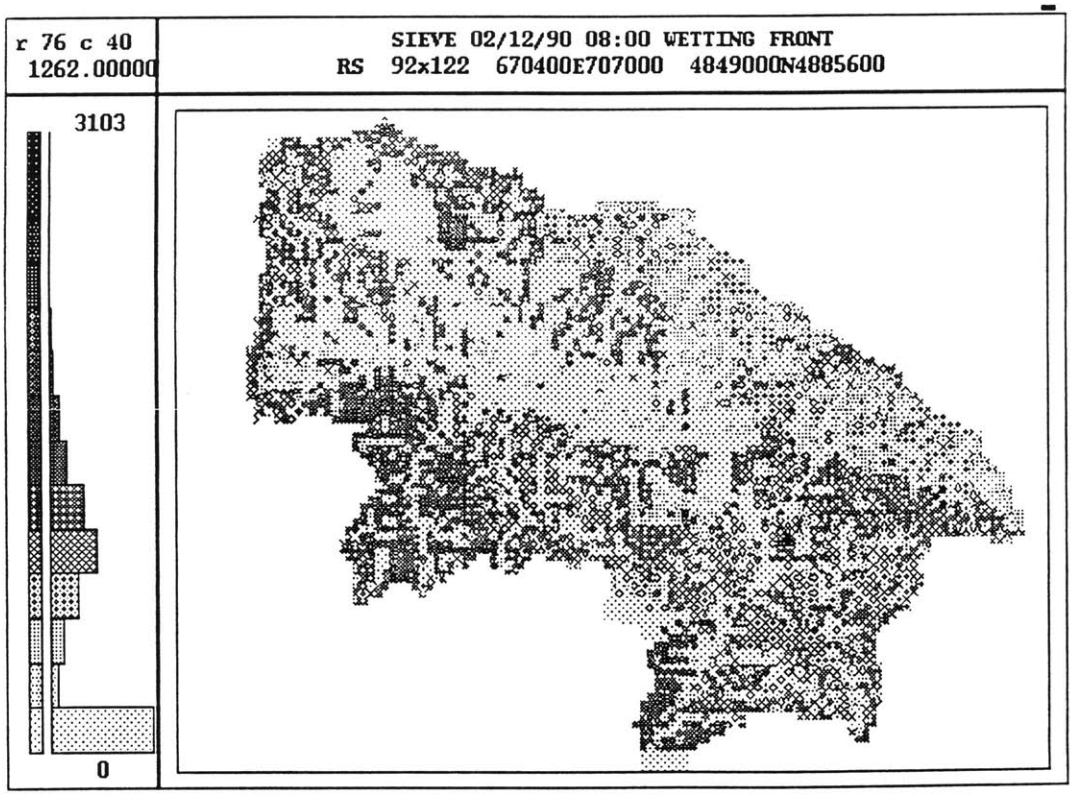


Figure 4.8: *Example of a basin display*

CHAPTER 5

Software Design of the Distributed Basin Simulator

This chapter is dedicated to the discussion of the implementation of the distributed basin simulator within the framework of RIBS. Software is developed in *C* language for an engineering workstation running the UNIX operating system. An overview of the design of the system is presented first, highlighting system components and their behavior. Then, a detailed description is given of the high-level software components of the simulator. A detailed presentation of software implementation is given in Appendix 5.

5.1 Design overview

This section presents an overview of the different components of the software system that implement the distributed basin model. A brief description of the software development methodology is given first, based on the definition of software objects as elementary building blocks of the system. The design process combines software and hydrologic model development, in a sense that the final result is not an application which satisfies a list of goals, but a software environment that can be used and manipulated in a number of ways to satisfy concrete requirements. In the

second part of this section, the structure and behavior of the four main software objects that implement the distributed model are presented. Software objects coincide with physical objects, meaningful to the hydrologist. The building blocks are self-contained, reusable and interchangeable. The software objects are shared by several modules, and software development takes care of maintaining multiple compatibility with all of them.

5.1.1 Programming methodology

System design followed the object-oriented methodology. The term "object-oriented" identifies a design and programming methodology centered around the notion of object, as opposed to other methodologies, such as structured programming, traditionally centered around the notion of function (Booch, 1991).

An object-oriented design approaches the solution to a problem through the definition of a set of objects which are structured as a hierarchy of entities sharing common properties and behaviors through inheritance. Objects are internally characterized by their data structure, which is private, and externally characterized by their behavior, which is public. The data structure is a list of variables or other objects. The behavior of an object is a list of functions which operate on its data structure. For instance, the object *Matrix* has a data structure containing information about dimensions and values. Its behavior would include operations such as *sum*, *multiplication*, *rank*, *transpose*, etc. In an object-oriented environment, both aspects, structure and behavior, are intimately coupled forming a unique entity.

Object-oriented techniques are usually associated with many specific terms, such as class, message, data abstraction, encapsulation, inheritance, etc., but two basic notions are crucial for the idea of object-oriented programming (Stefik and Bobrow, 1986):

- The idea of building a large system as a collection of individual entities with clearly defined boundaries to isolate them and standard protocols for communication. We use the generic term "encapsulation" to refer to this idea, although other words such as "data abstraction" are also related to it.
- The idea of structuring those entities in such a way that new objects can be derived from existing ones, specifying only the differences. This concept is usually called "inheritance". For instance, the notion of *SquareMatrix* can be derived from the notion of *Matrix*, specifying restrictions on the dimensions and adding new operations, such as *inverse*, *trace*, *eigenvalues*, etc.

The object-oriented approach in software development implements the classical "divide and conquer" strategy to overcome in part the inconveniences inherent to large software systems (Cox, 1986). The idea of the decomposition of a big system into smaller parts has been around in the software development community for many years under several different approaches. The main difficulty identified in practice for these design techniques has been the fact that a reduction in the size of elementary tasks to be performed is almost inevitably accompanied by a parallel increase in the complexity of the interactions among them. This is the problem addressed by the object-oriented approach. The division strategy is centered around "objects", as opposed to more traditional

software development techniques, where the division strategy is centered around functions or procedures. Rather than defining functions which call each other passing data as arguments, object-oriented methodologies define objects which interact sending messages to each other. Each object responds to a message performing a certain function included in its behavior. The main goal is to keep the interactions among the individual components (objects) under a strict control through protocols for message passing.

To design an object means defining a set of data together with some operations which can access those data and provide information to the rest of the application. The object integrates both data and procedures, as opposed to the traditional approach of treating data and procedures separately. Data contained in an object are private. They can only be accessed through the operations previously specified to read or modify them, so that unexpected side effects are kept under control. These operations are the effective channels of communication among the different objects which conform the entire system. Although objects may correspond to abstract notions, the objects considered in RIBS have physical meaning, and the operations defined for them correspond to their real behavior in the physical world.

The flow of information among objects is implemented through "messages". A message can be understood as a function call. Whenever an object in the application needs to interact with another one, it sends a message to it. The recipient object responds to the message performing the operation requested. The advantage of this approach is that the caller does not have to know anything about the particular implementation details of the object it is dealing with (except, of course, that the

functionality requested is in fact implemented for that object). An application can obtain the inverse of a matrix just passing the message "inverse" to the matrix. The implementation details of any particular object are encapsulated inside its internal structure and are only accessible to the caller through the object's specific procedures.

The objects of an application can also be structured into a hierarchy of classes. Classes are the unit of modularity in an object oriented system. A class describes the generic characteristics of a group of objects. Individual objects in the application are particular instances of the abstract concept represented by the class. The relation between a class and an instance is similar to that between the concept 'river basin' and the Sieve catchment. The operation of defining an instance of an object is called "instantiation". It can be seen as defining a variable of a given type. An instance *matrix_a* of the class *Matrix* is an area in the memory which stores a data structure of the type *Matrix*, just as the variable *foo* can store a double precision floating-point data structure. The significant difference is that the definition of *Matrix* also includes a list of operations that know the data structure and operate on it.

From the standpoint of software engineering, the main benefit of the object oriented-programming paradigm is the potential for software reusability. Encapsulation provides a mean of isolating the objects within a system, so that if they change, other parts of the system can remain unchanged as long as the previous functionalities are maintained. In principle, the internal organization of the *Matrix* class can be changed without having to recompile all the applications using it.

From the standpoint of physically-based modeling, object-oriented design provides a way of mapping the hydrological analysis of the problem into the software design of the application. A natural design identifies the components of the physical system with the software objects in the application. Great potential benefits can be obtained from the identification of the entities manipulated by both processes, hydrologic analysis and software design. Modularity also allows for several approaches to the same hydrologic problem from different points of view, manipulating differently the same concepts, represented by software objects. Object-oriented design provides a mean of sharing common data structures and functionalities among objects which have similar characteristics.

The functional units discussed in Section 4.4.1 were the functional design objectives for the distributed basin simulator. According to the functional analysis, four main objects were identified as high-level

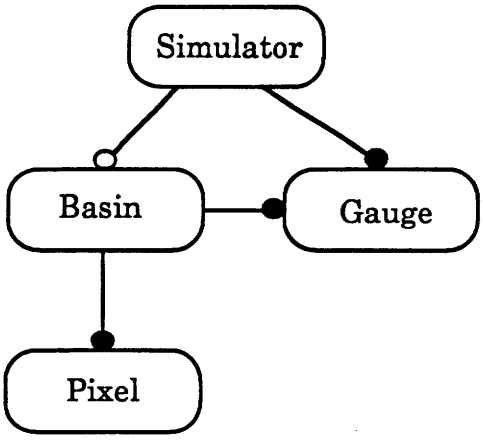


***Figure 5.1**: High-level components of the distributed basin simulator*

components of DBS: the *Pixel* object, the *Basin* object, the *Gauge* object and the *Simulator* object. Other objects are also necessary in the design as auxiliary data structures with specific functions, but these four objects constitute the core of DBS's architecture. Figure 5.1 shows the mutual relations among them. The *Pixel* object represents a soil column as that described in Chapter 2, and performs the local runoff generation. The *Basin* object represents the collection of individual pixels as a whole, and takes care of the spatial interactions in runoff generation. The *Gauge* object controls surface water transport in the basin. The *Simulator* object controls global model inference and is responsible for input/output operations.

5.1.2 The *Pixel* object

The *Pixel* object implements the behavior of the soil column. Figure 5.2 represents its object diagram, including internal variables, related objects and types of methods defined for it. The notation for the object diagram follows the conventions proposed by Rumbaugh et al. (1990). Objects are represented by a round-cornered rectangle with class denomination on top of it. The upper half of the rectangle describes the data structure and the lower part of the rectangle describes object behavior. A hollow circle represents a one-to-one relationship, and a solid circle represents a zero-or-more relationship, meaning that the parent object may have none, one or more children of that type.

Data structures

The *Pixel* object uses two auxiliary data structures: the *StVars* object and the *GlobData* object. The *StVars* object contains information about the state variables of the model, and is the core of pixel representation. The *GlobData* structure actually belongs to the *Basin* object, but it is also made available to the *Pixel* object, It contains information about pixel properties which are uniform throughout the basin.

The *Pixel* structure contains pointers to state variables and global data structures and other variables regarding the soil column. The variables in the *Pixel* structure are of two types: pixel properties and deduced variables. Pixel properties represent static attributes of the pixel

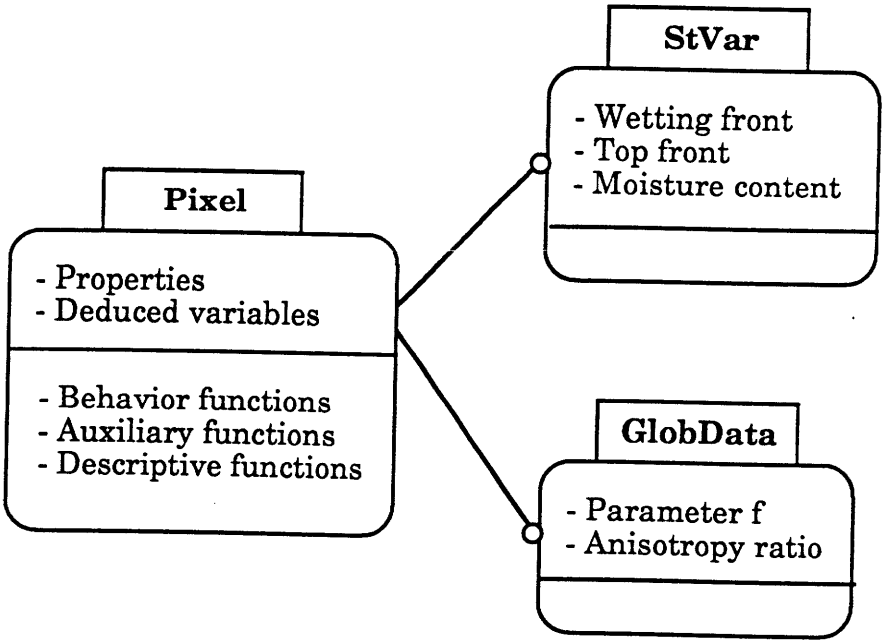


***Figure 5.2**: Structure of the Pixel object.*

(values which do not change during pixel evolution), such as soil properties or topographic and geometric characteristics. Deduced variables represent dynamic attributes of the pixel. They are either boundary conditions, imposed by the neighboring pixels or the external forcing, or derived values that can be obtained from the state variables and the static properties. This last group would be more naturally represented in an object oriented environment through return values of behavior functions, but their inclusion in the *Pixel* data structure is justified by reasons of computational efficiency.

Object behavior

The behavior of the *Pixel* object is represented through its member functions. *Pixel* behavior implements the one-dimensional model of infiltration described in Chapter 2, and it is described in detail in Section 5.2. Here we present an introduction of the groups of functions included. There are three types of member functions in the *Pixel* object:

-*Behavior functions*: This group of functions implements the evolution of the pixel during a computation time step. Behavior functions do not return any value, but they generate side effects. They operate on the pixel data structure modifying the values of one or several internal variables. Examples of behavior functions are the computation of front evolution or moisture balance in the soil column.

- *Auxiliary functions*: Auxiliary functions perform basic operations needed by behavior functions. They refer to entities whose computation involves enough complexity to justify a function call, but whose use is not regular or frequent enough to justify a variable. They return a value, but

they do not have any other side effect. Examples of auxiliary functions are those that return the moisture content at a given depth, the equivalent rainfall rate, the saturation level for a given rainfall rate, etc.

- *Virtual variables*: Virtual variables represent attributes that can be directly obtained from the state of the pixel. This group of functions is mainly intended for the user interface, and their number and complexity is a function of user needs. Virtual variables return a floating point value and do not have any side effects. Examples of virtual variables are the surface infiltration capacity, the degree of saturation or the moisture deficit until saturation.

5.1.3 The *Basin* object

The *Basin* object represents the aggregation of individual pixels that conform the river basin. A schematic representation of the structure of the *Basin* object is shown in Figure 5.3.

Data structures

Auxiliary objects for the *Basin* object are: *GlobData*, *SoilData*, *Map* and *BasinTrace*. The *Basin* object stores global basin variables, the values of the input, output and state variables of individual pixels, and information about the topological relationships among them. It also contains the necessary information to provide trace information at different levels for debugging purposes. These different data groups are briefly commented hereafter.

The description of basin properties is represented in two data structures: a *GlobData* structure and a list of *SoilData* structures. *GlobData* contains global attributes which are assumed constant throughout the basin, such as routing velocities or anisotropy ratio. Attributes can be transferred easily between the *GlobData* group and the list of distributed variables, since its definition is encapsulated in the *Basin* object. This gives flexibility to the basin representation. *SoilData* contains values of soil properties for a given soil type, such as saturation

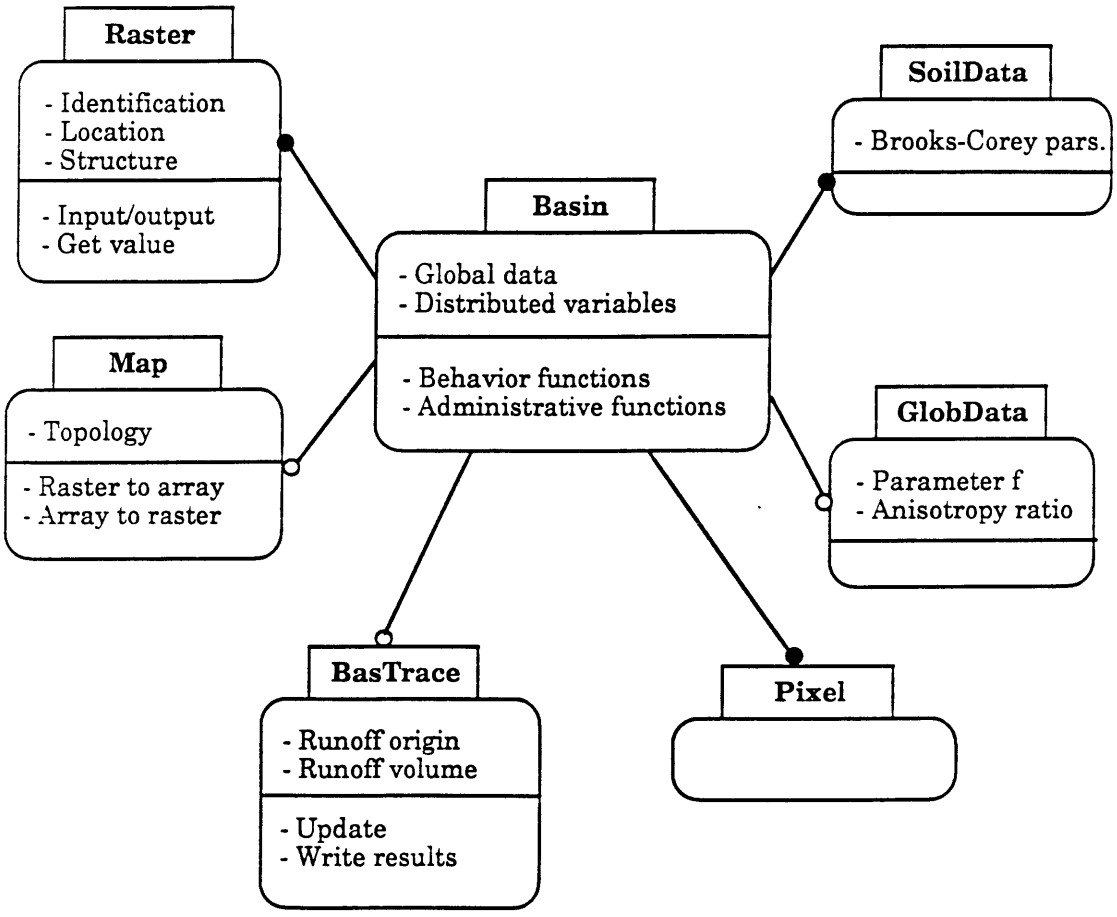


***Figure 5.3**: Structure of the Basin object.*

moisture content or hydraulic conductivities. The assumption that soil characteristics of individual pixels can be grouped in soil types is made, and therefore only a limited number of soil types is needed to represent all pixels in the basin. The information about different soil types is contained in the *SoilData* structure within the *Basin* object.

The *Basin* object also groups distributed information about individual pixels. In a pure object-oriented environment, the *Basin* object would manage a list of *Pixel* objects, each one representing a single pixel element. However, the large number of pixels in a particular basin means large memory requirements to handle objects representing individual pixels. Since pixels can be processed individually, the solution of keeping only one *Pixel* object was adopted. During the computation loop, the *Pixel* object gets successively reinitialized with values corresponding to particular pixels. Permanent pixel properties and state variables are stored in the *Basin* object in the form of arrays. In order to optimize memory requirements, the topological structure of the basin is stored in a *Map* object. The map keeps track of the spatial distribution of pixels and provides utility functions to relate row and column index in the grid with pixel position in the arrays.

Trace functions are performed by the *BasinTrace* object. The *BasinTrace* object stores detailed information about how runoff generation is distributed throughout the basin, both in terms of number of pixels in every possible runoff-generating state and volumes of runoff generated by pixels in every state. Since tracing is expensive in terms of resource and computational requirements, the creation and processing of the trace object is optional.

Object behavior

The *Basin* object performs two basic functions: it manages storage and retrieval of information regarding basin characteristics and state, and evaluates the distribution of runoff generation over the basin. Auxiliary functions are also available for the *Basin* object to initialize the *Pixel* object, create trace objects and for general memory management. The related objects (*Map*, *BasinTrace*) also perform specific functions

The information management functions are in charge of storing and retrieving information regarding individual pixels. They basically initialize the *Pixel* object for every soil column in the basin and store the results offered by the *Pixel* object in the *Basin* data structure. The runoff generation behavior of the *Basin* object is decomposed in two member functions: *para_flow_loop()* and *hill_loop()*. *para_flow_loop()* implements the computation of moisture transfer between pixels due to lateral disequilibrium of moisture content. *hill_loop()* implements the runoff generation loop, including moisture transfer between elements in homogeneous terrain. Both functions loop over the individual pixels in the basin, initializing the *Pixel* object and applying the pertinent member functions. The *BasinTrace* object offers optional update functions and result dumping.

5.1.4 The *Gauge* object

The information about routing is stored in *Gauge* objects. A gauge represents a point in the basin where discharges are to be computed. The basin usually contains several instances of the *Gauge* object, one for every

point in which information about streamflow is required. Figure 5.4 shows the internal structure and behavior of the *Gauge* object.

Data structures

The *Gauge* data structure contains basic information about the gauge, such as location of the outlet, denomination and maximum distance of travel, and three auxiliary data structures: *RoutingMap*, *Results* and *GaugeTrace* objects. The *RoutingMap* object is a data structure that contains the topological information of the contributing

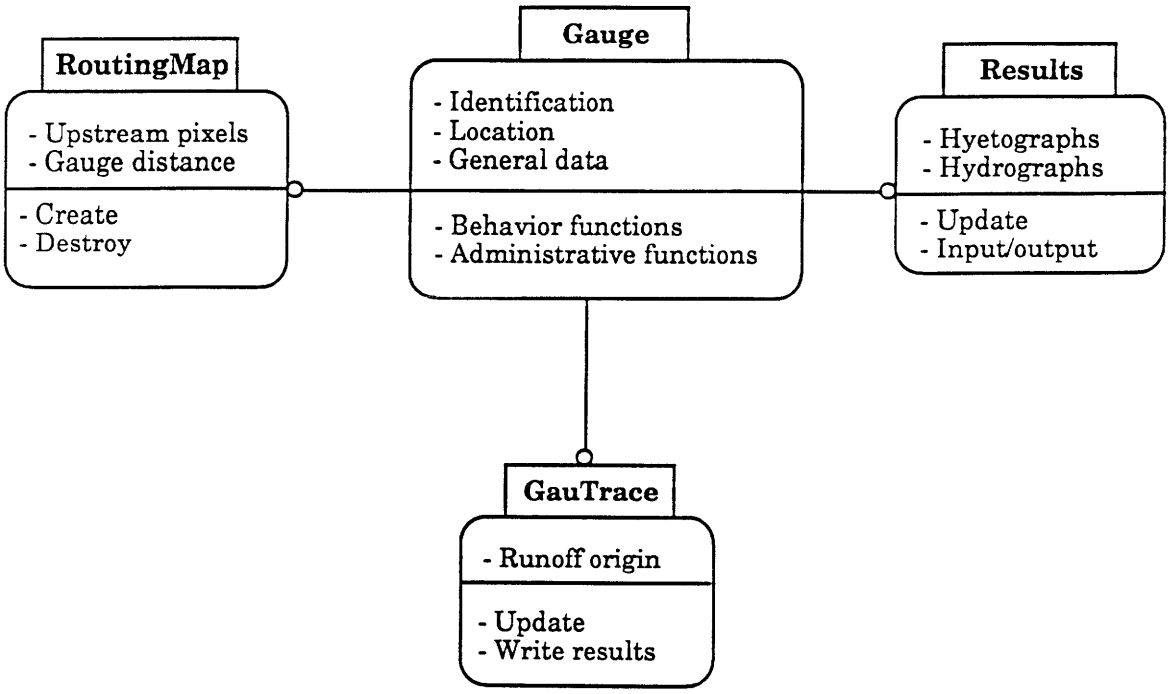


***Figure 5.4**: Structure of the Gauge object.*

area to the gauge location. The *Results* object stores several hyetographs and hydrographs corresponding to the area upstream of the gauge location. The *GaugeTrace* object provides additional information about how streamflow is generated: time evolution of the different modes of runoff generation.

Object behavior

The *Gauge* object implements the distributed convolution operation for the catchment upstream the gauge location. It uses the runoff generation information stored in the *Basin* object for every pixel and routes it to the outlet. That basic functionality of the *Gauge* object can be used by any client object. For instance, the *Simulator* object need only to apply the routing function at high level for every instance of the *Gauge* object in the basin. That is equivalent to routing for each gauge point independently. The user interface can offer custom-made hydrographs at any location within the basin because it only has to create a *Gauge* object at that location and use its routing functionality. *Gauge* also implements other functionalities needed by the user interface, such as unit hydrograph evaluation or routing of individual pixel responses. The *Gauge* object also creates and destroys instances of *RoutingMap*, *Results* and *GaugeTrace* as needed. The *GaugeTrace* object can be activated to produce a decomposition of runoff in different modes.

5.1.5 The *Simulator* object

The *Simulator* object deals with the high-level managing of simulations using the functionalities provided by the *Basin* and the *Gauge* objects. In addition to governing the real-time operational loop, the simulator is in charge of keeping track of time control and of performing the input/output operations. Figure 5.5 shows an schematic representation of the *Simulator* object.

Data structure

Auxiliary data structures for the *Simulator* object are a *Timer* and an input/output data structure (object *IOdata*). The *Timer* objects includes

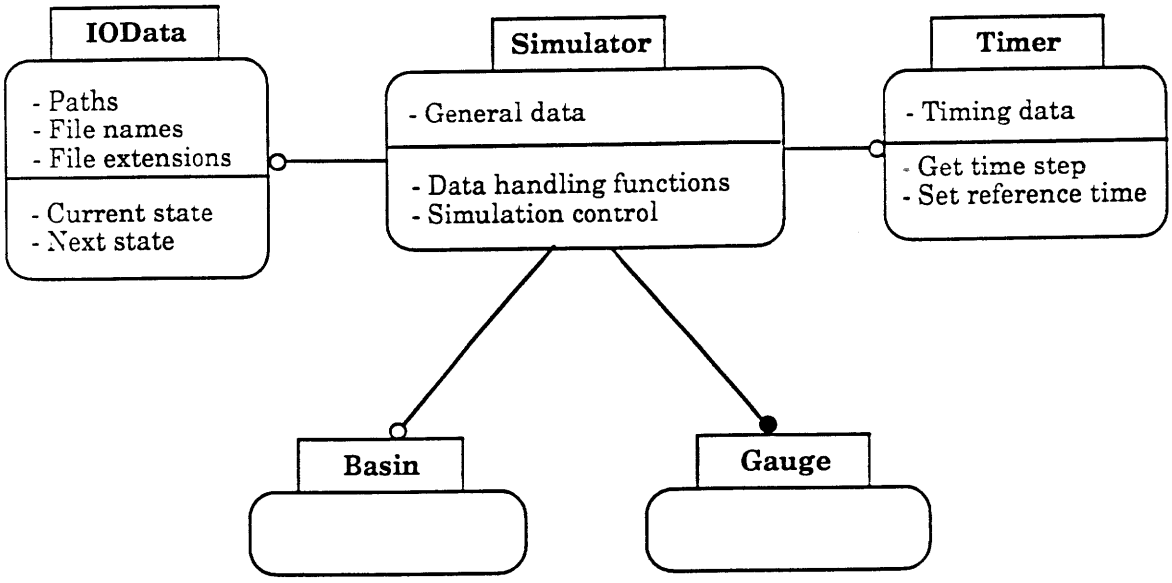


***Figure 5.5**: Structure of the Simulator object.*

information about start and end time of the simulation, computation time increment, result time resolution, current time, etc. The input/output data stores information about paths and file names for the different types of data involved, number and locations of gauges, special settings for output display in the user interface and file pointers for reading and writing.

Object behavior

The behavior of the *Simulator* object covers two basic functionalities: data handling and simulation control. The data handling operations cover file reading and writing and interactions with the other elements of the real-time system (communications with the rainfall generation module and with the real-time interface). The read and write operations are decomposed into several independent functions which may be called from different contexts, thus providing flexibility to the scheme.

The simulation control refers to the management of the operational loop. The three functional units related to basin simulation: computation loop, rainfall loop and forecasting loop, are the basic tasks of the *Simulator*. The simulator uses the functionalities of the *Basin* and *Gauge* objects to simulate basin evolution according to the rainfall information received. Since the duration of rainfall intervals is not specified a priori, the simulator operates in an open loop, making decisions about reading and writing information according to the information received from the rainfall control module. The simulator is also in charge of creating and controlling its children objects, such as *Basin* and *Gauge*, reserving and releasing memory for them.

5.2 Implementation of the 1-D model of infiltration

The one-dimensional kinematic infiltration model presented in Chapter 2 is implemented in the *Pixel* object. The *Pixel* object is the core of the DBS application, since it carries out the most significant part of model inference. The *Pixel* object represents the state of a soil column, and it is the local evolution of soil state that controls the patterns of runoff generation throughout the basin. Furthermore, since the number of pixels in a basin is typically of the order of several thousands, implementation details of the *Pixel* object are extremely important in terms of overall model efficiency. This section presents a discussion of the solutions adopted to implement the basic functional unit of the application, with special attention focussed on the numerical scheme used to integrate model equations and the treatment of potential model instabilities.

5.2.1 Methods of the *Pixel* object

The methods of the *Pixel* object can be classified in three main groups. The functional unit assigned to the pixel object, the one-dimensional model of infiltration, is implemented in the behavior functions group. Other model equations of internal interest to the *Pixel* object are grouped as auxiliary functions. The third group consists of descriptive functions that are used by the user interface to offer information about pixel state through the concept of virtual variable. We only discuss behavior functions in detail, since the implementation of the others is straightforward.

Behavior functions

Behavior functions are those that implement the one-dimensional kinematic model of infiltration for the pixel. They are offered to client objects as external images of pixel behavior. A external client is only interested in the apparent behavior of the pixel, such as runoff generation or front position, but not in internal details of pixel evolution, such as front speed. Behavior functions deal with the implementation of front and moisture dynamics for the pixel and with the verification of model results, checking for consistency among the values of all variables at the end of the computations. The final outputs are the new values of state variables in the pixel and its runoff generation. The implementation of behavior functions is discussed in Section 5.2.2.

Auxiliary functions

The auxiliary functions provide convenient computation of several magnitudes. They are intended for use only by the pixel object, not by external clients. They are usually straightforward implementations of model equations or formulas, which are defined as functions because they appear frequently in different contexts. Examples of auxiliary functions are *Re()*, which returns the equivalent rainfall rate for a given unsaturated moisture content, *Mu()*, which returns the unsaturated moisture content, *Theta()*, which returns moisture at a given level, etc.

Descriptive functions

Given the parameterizations adopted by the model to describe the vertical distribution of moisture, only three variables are needed to define pixel state at any given time: position of the top and the wetting fronts and moisture content. Descriptive functions offer derived magnitudes that can be inferred from the value of the three state variables plus the intensity of runoff generation. They are typically used by the user interface to offer information about pixel state. Variables such as the degree of saturation of the soil column or the infiltration capacity at the surface can be offered to the user as a clarification of the practical implications of pixel state at any given time.

5.2.2 The one-dimensional model of infiltration

The main behavior function of the *Pixel* object is *hillpix()*, which computes pixel evolution and runoff generation for a soil column, following the model presented in Chapter 2. Model equations are formulated in terms of three state variables: wetting front position, top front position and moisture content. Evolution equations are available for the three state variables. This discussion deals with the specific problems that arise when a numerical integration of model equations is attempted. Two important issues arise in the basin-scale implementation of the model. First, the solution of considering an equivalent rainfall rate to deal with variable rainfall rates has important implications on the numerical stability of the model. Second, the spatial patterns of drainage in a river basin lead to subsurface flow accumulation in many points. For those grid

elements that represent points of aggregation of subsurface flow, the inflow is usually larger than the outflow, and the moisture evolution equation may be dominated by the subsurface term. If that is the case, the numerical stability of the model may also be affected, because it was conceived for the case in which the vertical infiltration is dominant.

Model equations are solved in three consecutive steps. In the fist step (function *front_evolution()*), front dynamics are solved using an explicit finite difference scheme. State-variable values at the beginning of the time step are used to evaluate the coefficients in the equations, and proposed values for the positions of the fronts at the end of the time step are obtained. In the second step (function *moist_evolution()*), moisture balance is performed, using the updated front positions to evaluate local infiltration and subsurface outflow. The result is the new value of the moisture content of the pixel. In the third step (function *comp_runoff()*) the compatibility of front positions with the new moisture content of the soil column is tested and the runoff generation in the pixel is evaluated.

Front dynamics

Front evolution equations are integrated using an explicit finite difference scheme. The coefficients of the equations are evaluated using the values of the state variables at the end of the previous time step. If the integration time step is Δt, the value of the wetting front position for unsaturated infiltration at time $j+1$ is given by

$$N_f^{j+1} = N_f^j + \frac{(R_e^j - R_i)\cos(\alpha)}{\theta(R_e^j, N_f^j) - \theta(R_i, N_f^j)} \Delta t \qquad (5.1)$$

where the superscript represents the time step at which variables are evaluated.

For saturated infiltration, two similar equations represent the integration scheme for the wetting and the top front:

$$N_f^{j+1} = N_f^j + \frac{q_n^j - R_i \cos(\alpha)}{\theta_s - \theta(R_i , N_f^j)} \Delta t \qquad (5.2a)$$

$$N_t^{j+1} = N_t^j + \frac{q_n^j - R_e^j \cos(\alpha)}{\theta_s - \theta(R_e^j , N_f^j)} \Delta t \qquad (5.2b)$$

where q_n is the normal infiltration in the saturated area, evaluated as

$$q_n^j = K_{0_n} \frac{f(N_f^j - N_t^j)}{e^{fN_f^j} - e^{fN_t^j}} \cos(\alpha) \qquad (5.3)$$

This integration scheme leads to very good approximations under most conditions, since state variables change very slowly during a storm. There are circumstances, however, under which the explicit scheme is inaccurate or unstable, and additional precautions have to be adopted.

The explicit numerical scheme is inadequate when the normal gradients of the equivalent moisture or the initial moisture content are large. The unbalance between normal infiltration flow above and below the front is translated into front advance via the difference between moisture contents below and above front position at the end of the previous time step. In reality, it is the integral of the difference between moisture

functions what should be taken into account. If that difference varies considerably from the initial to the final position of the front during the time step, the approximation given by the numerical scheme is poor.

These situations are likely to occur when either the equivalent or the initial moisture content are close to saturation, since the gradient of the moisture function grows with depth. That translates into a very small denominator in Equations (5.1) or (5.2) that leads to numerical instabilities. Model implementation checks for that situation and applies alternative numerical schemes when that occurs. The case in which saturation develops in the pixel can be solved explicitly. When $N_f^{j+1} > N^*(R_e^j)$ saturation develops in the pixel. In this case, the evolution of N_t may be obtained by taking the limit in Equation (2.17b) when N_t approaches N_f and N^*. Substituting $\theta(R,N_t)$ and q_n by their respective expressions (Equations 2.7 and 2.12) and taking limits in (2.17b) we obtain

$$\lim_{N_t \to N^*} \frac{dN_t}{dt} = - \frac{\varepsilon R \cos(\alpha)}{\theta_s - \theta_r} \qquad (5.4)$$

Other cases of small denominator are solved iteratively. The scheme applied is based on substituting $\theta(R_e^j, N_f^j)$ on Equation (5.2a) by the average $\frac{\theta(R_e^j, N_f^j) + \theta(R_e^j, N_f^{j+1})}{2}$. The resulting implicit expression is solved iteratively until a satisfactory value for N_f^{j+1} is obtained. Since only a few pixels are in this situation, the overall computational efficiency of the scheme is not affected by the iterative solution. A small denominator in Equation (5.1) can also occur when R_e is very close to R_i, but in this case the numerical scheme is adequate because both functions are similar along all the profile.

After front evolution has been computed, additional checks are performed to verify two basic constraints. First, the wetting front should not get deeper than the water table. Normal infiltration capacity below the water table is effectively null, because the water table position is in equilibrium with inter-storm subsurface saturated flow. Therefore, when the wetting front reaches the water table its position remains constant and the infiltration flow below it is now zero. The net effect is a local rise in the water table (the saturated area) in the pixel, since moisture now accumulates above the wetting front. The evolution equations used when the wetting front reaches the water table are

$$N_f^{j+1} = N_f^j \tag{5.5a}$$

$$N_t^{j+1} = N_t^j - \frac{R_e \cos(\alpha)}{\theta_s - \theta(R_e, N_t^j)} \Delta t \tag{5.5b}$$

The second check deals with the relative position of N_t and N^*. N^* is the level at which the equivalent moisture profile R_e reaches saturation, and is given by Equation (2.9). In the original model formulation the rainfall intensity is a constant, and therefore front evolution equations are obtained considering that only N_f and N_t change with time (see Cabral et al., 1990). If R is substituted by an equivalent rainfall intensity R_e to deal with variable rainfall, model equations should in theory account for $\frac{dR_e}{dt}$. That is extremely difficult in practice, since no analytical expression is available for R_e. However, R_e changes very slowly during the storm, and $\frac{dR_e}{dt}$ has little effect of front evolution and is usually negligible.

Nevertheless, there is one case under which the variation of R_e has important consequences, and that is when N_t is near N^*, because the application of the explicit scheme to obtain the evolution of N_t may lead to a position of N_t incompatible with the value of R_e in the next time step. N_f, N_t and N^* must always satisfy the relation: $N_f \geq N^* \geq N_t$. Lack of accuracy in the numerical scheme may lead to a value for $N_t^{j+1} < N^*(R_e^{j+1})$. Therefore, the value obtained for N_t must be checked to correct this inconsistency.

The function *front_transition()*, which computes the values *incr_Nf* and *incr_Nt* (rates of change of N_f and N_t during the time step) implements the numerical scheme described above. The function differentiates five basic pixel states. Interesting situations arise whenever the pixel changes from one state to another, since state transitions are usually associated with numerical instabilities. The function first verifies pixel state, and applies the equations corresponding to that state to obtain front evolution. It also controls the possible state transitions to guarantee numerical stability. The actions for every state are as follows.

-*Unsaturated pixel*: It is characterized by N_f=N_t. The evolution of both fronts is evaluated considering Equation (5.1). Transitions out of this state may be due to three factors. The most usual transition is to the saturated pixel state, and is marked by N_f becoming greater than N^*. The second transition represents the emptying of the pixel, and is marked by R_e becoming less than R_i. In this case, the state variables are reset to their initial values. The third possible transition occurs when the wetting front reaches the water table. If the water table is located at the saturation level for the initial moisture content, this transition is not possible, since the pixel should become saturated first.

- *Perched-saturated pixel*: In this situation $0 < N_t < N_f$. Governing equations for front dynamics are (5.2a) and (5.2b). Transitions out of this state may lead to any other state. Transition to the unsaturated state occurs when N_t becomes equal to N_f. Transition to the surface-saturated state occurs when N_t reaches the surface. The pixel may also reach the deep-saturated state if the wetting front reaches the water table before the top front reaches the surface. This transition is also numerically unstable, since the numerator of Equation (5.2a) becomes zero. A numerical stability check is performed in this situation. Finally, although it is extremely unlikely, this pixel may evolve to a fully-saturated state if the top front reaches the surface and simultaneously the wetting front reaches the water table.

- *Deep-saturated pixel*: This situation is characterized by $0 < N_t < N_f = N_{wt}$. Front evolution is governed by Equations (5.5.a) and (5.5b). The pixel may evolve to a fully-saturated state when the top front reaches the surface or to an unsaturated state if the top front reaches the water table. If the initial moisture content leads to saturation at the water table, this transition means that the pixel has dried out, and therefore the state variables are reset to 0.

- *Surface-saturated pixel*: A pixel is surface-saturated if $0 = N_t < N_f < N_{wt}$. Front evolution is governed by Equations (5.2a) and (5.2b), applying $\frac{dNt}{dt} = 0$ in case Equation (5.2b) gives negative front velocity. Possible transitions are to the perched-saturated state if $\frac{dNt}{dt} > 0$ in Equation (5.2b) or to fully-saturated state when N_f becomes grater than N_{wt}.

- *Fully-saturated pixel*:In this situation $0 = N_t > N_f = N_{wt}$. Unless moisture outflows are greater than moisture inflows, front situation is

static. The only possible state transition is to a deep-saturated state, when N_t becomes greater than zero.

Moisture balance

Once front evolution has been computed, the equation for moisture evolution in the pixel is solved in the function *mois_evolution()*. The updated front positions are used to obtain moisture inflows into and outflows from the pixel, in order to evaluate more accurately the moisture balance. Moisture evolution is given by the numerical approximation of Equation (2.33):

$$M_s^{j+1} = M_s^j + \left(R_{inf}^{j+1} + \frac{Q_{in}^{j+1} - Q_{out}^{j+1}}{A} \right) \Delta t \qquad (5.6)$$

R_{inf} is the infiltration rate, given by the comparison of rainfall intensity in that time step with infiltration capacity. Infiltration capacity is obtained as a function of surface hydraulic conductivity and front position, according to Equations (2.28), (2.29) and (2.30). Q_{in} is the sum of subsurface inflows into the pixel, obtained as a result of computations for the upstream pixels. Q_{out} is the subsurface outflow from the pixel, obtained as a function of front position applying Equation (2.20). The only numerical check that needs to be done is the verification that the storm moisture content is positive. If outflows are larger than inflows for a given time step, moisture content decreases during that time step. If moisture becomes negative it means that the pixel has dried out. In that case, the value of Q_{out} is corrected and the pixel is reset to the initial condition.

Runoff evaluation

The last step in pixel computation is the evaluation of runoff, implemented in the function *comp_runoff()*. Two types of runoff are considered: Hortonian runoff and return flow. Hortonian runoff is obtained comparing rainfall intensity and infiltration capacity. Return flow is generated if inflows into the pixel exceed the water holding capacity of the soil column. If the total moisture content of the pixel is higher than that corresponding to total saturation above the wetting front, return flow is generated

Total moisture content M_t above the wetting front is the sum of the moisture content corresponding to the initial condition M_i and the storm moisture content M_s. M_i is given as the integral of the initial moisture profile from N_f to the surface

$$M_i = \int_0^{N_f} \theta(R_i , n)\, dn \qquad (5.7)$$

and M_s is given by Equation(5.6). The maximum water-holding capacity of the soil column is $M_{max} = \theta_s N_f \cos(\alpha)$. Therefore, return flow is given by

$$R_r = M_t - \frac{\theta_s N_f}{\cos(\alpha)} \quad \text{if} \quad M_t > \frac{\theta_s N_f}{\cos(\alpha)} \qquad (5.8a)$$

$$R_r = 0 \quad \text{if} \quad M_t \le \frac{\theta_s N_f}{\cos(\alpha)} \qquad (5.8b)$$

If return flow is generated, the moisture content is set to

$$M_s = \frac{\theta_s N_f}{\cos(\alpha)} - M_i \qquad (5.9)$$

State verification

The algorithm described above works well for most conditions. There are, however, certain cases for which the consecutive application of front evolution and moisture balance equations leads to inconsistent results. These situations are frequent in pixels with shallow water table or high anisotropy ratio. A number of verifications are performed in the last step of the pixel response function. Most problems are usually related with the equivalent rainfall rate. After the new front positions and moisture content have been evaluated, it must be checked that $R_e > R_i$ and that $N_t < N^*$.

R_e is obtained as a redistribution of the moisture content in the unsaturated area M_u. M_u is the moisture above the wetting front, and it is given by

$$M_u = M_s + M_i - \frac{\theta_s (N_f - N_t)}{\cos(\alpha)} \qquad (5.10)$$

Re is then given by Equation (2.19). If the resulting R_e is smaller than R_i, the actual moisture distribution in the pixel does not coincide with that assumed by the front evolution equations. This situation may appear in pixels with high anisotropy ratio, for which the dynamics of the pixel are dominated by subsurface lateral flow, rather than by normal infiltration.

If the saturated profile (N_f - N_t) is large compared to N_t the subsurface outflow from the pixel may be significant. Subsurface outflow does not affect directly the position of the fronts; only in the next time step will the decreased moisture content affect the front evolution equations. Therefore, it is possible to find a perched-saturated pixel with a large saturated area that finishes the time step with an empty unsaturated area. The solution adopted in this cases is a compromise. The top front position is lowered in order to extract moisture from the saturated area and fill in part the unsaturated portion of the soil column. Since only an extremely small fraction of the pixels during all time step are in this situation, this solution does not affect the overall behavior of the basin, and is only used to maintain the simulation within physically meaningful limits.

Another possibility is that the evolution of the moisture content lead to an excessive moisture in the unsaturated area. This is the opposite of the situation discussed previously. Here subsurface inflow is the disturbing factor, and the consequence is that the moisture content in the unsaturated area cannot be accommodated above the top front. That is equivalent to having N^* (saturation level for R_e) above N_t. The solution adopted is to move up the position of the top front until a value of M_u is found that satisfies the restrictions. A trial and error iterative procedure is used to get the new value of N_t.

5.3 Implementation of basin response

The simulation of basin response in the Distributed Basin Simulator is divided in two independent procedures: runoff generation and flow

routing. This section deals therefore with the implementation of the two functional units: basin-scale runoff generation and distributed convolution, related to the mentioned procedures. The first one is associated with the *Basin* object and the second one is associated with the *Gauge* object.

5.3.1 Implementation of runoff generation

The *Basin* object implements the evaluation of runoff generation at the basin scale. The basin is an organized collection of pixels, related through the link 'drains to'. The behavior of individual pixels is implemented in the *Pixel* object, and therefore, all the *Basin* object has to do is apply the functionalities of the *Pixel* object and take care of the connectivity between pixels. Two kinds of methods are of interest in the *Basin* object: administrative functions, which are necessary for management of children objects (creation, destruction, memory allocation and release, etc.) and behavior functions, which implement the basic functionalities for client objects.

Administrative functions

The functions *init_pixel()* and *end_pixel()* are the links between the *Basin* and the *Pixel* objects. A basin is typically composed of several thousand pixels, and therefore it is impractical to create a *Pixel* object for every grid node in the basin, because memory requirements are excessive. The solution adopted is to maintain only as many *Pixel* data structures as are simultaneously necessary to perform the basic basin operations, and

use member functions of the *Basin* object to initialize and terminate the instances of the *Pixel* object. The functions *init_pixel()* and *end_pixel()* are used to transfer information between the *Basin* and the *Pixel* data structures. *init_pixel()* takes the information corresponding to a particular pixel in the basin and stores it in the variables of the *Pixel* data structure. Once the pixel has been initialized, client objects can request from it the operations encoded in its behavior. The function *end_pixel()* can be used to store back in the *Basin* data structure the modified variables of the pixel: state variables and runoff generation.

Other administrative functions of the *Basin* object are those which control memory allocation for itself and for its child object, *BasinTrace*. The memory necessary to store the distributed variables in the basin is allocated in the function *bas_start()*, which should be called after the instance of *Basin* has been created, in order to obtain the size of the arrays. *init_bas_trace()* and *end_bas_trace()* create and destroy instances of *BasinTrace* objects.

Behavior functions

Basin behavior implements the spatial runoff generation mechanism in the basin. It uses the functionalities provided by the *Pixel* object to infer local runoff generation in individual pixels, but it must add functionalities to deal with spatial interactions between pixels, basically related to subsurface moisture transfer between pixels. Two types of interactions are of interest: lateral flows for homogeneous terrain and lateral flows resulting from the spatial variability. Modeling philosophy

and model equations have been described in Section 2.2.3. Here we deal only with the implementation details of model equations.

The two mechanisms considered for lateral flow generation are essentially different in the sense that one is a unidirectional process, whose order is imposed by the relation 'drains to' and the other is a bidirectional process in which two neighboring pixels mutually interact with each other. That means that lateral flow must be evaluated in two separated steps. Lateral flows for homogeneous terrain can be evaluated at the same time as the runoff loop is being evaluated, as long as upstream pixels are processed first, but lateral flows resulting from spatial variability must be evaluated in a loop in which all pixels are at the same stage of computation.

Two different functions are therefore needed to evaluate the two types of lateral flows. Lateral flow for homogeneous terrain is evaluated in a function called *hill_loop()*, which also evaluates pixel evolution. *hill_loop()* operates on all the pixels of the basin in an orderly way, starting for the most upstream pixels. It uses the routing map of the gauge number 0, which always corresponds to the total basin outlet, in which the pixels are arranged following the inverse of the recursive relation 'drains from'. *hill_loop()* calls *init_pixel()* to initialize the *Pixel* data structure, calls *hillpix()*, a method of the *Pixel* object, to evaluate pixel runoff and lateral flow, and stores the results calling *end_pixel()*. *end_pixel()* also accumulates the lateral flow resulting from the pixel into the lateral inflow variable corresponding to the pixel located downslope of it.

Lateral flows resulting from spatial variability are evaluated in the function *para_flow_loop()*. *para_flow_loop()* initializes the *Pixel* data

structure and evaluates the *Pixel* member function *outflow_complement()* for the pixel of concern and for the one located downslope of it, and takes the average as described in Chapter 2. *outflow_complement()* is just a straightforward implementation of Equation (2.27). Finally, *para_flow_loop()* stores the results as subsurface inflow for the downslope pixel.

The children objects, *Pixel*, *Map* and *BasinTrace* also have member functions. The behavior of the *Pixel* object was already discussed in Section 5.2. The *Map* object is the link between the *Raster* and the *Basin* objects. Distributed variables are stored as arrays in the *Basin* object. When these variables are stored in a file, the actual values are combined with topological information about the spatial location of every grid point, and the combination constitutes a *Raster* object, which can be stored in the database using the I/O functions available. The *Map* object stores information about the correspondence between locations in the *Raster* structure (identified by row and column) and positions in the basin arrays (identified by array index). Its methods *get_pos()* and *get_loc()* provide one piece of information given the other. The *BasinTrace* object keeps a detailed accounting of the origin of the runoff generated in the basin at every time step. It controls the number of pixels in every runoff-generating state and the actual volume of runoff generated in every mode. If the basin trace is activated, *hill_loop()* calls the function *tr_update()* to store information about every pixel in the trace structure.

5.3.2 Implementation of surface flow routing

Surface flow routing is implemented as a functionality of the *Gauge* object. The surface flow routing algorithm is applied in two main contexts. First, it is a necessary part of model inference, and it must be offered as a functionality to the *Simulator* object to be included in the computation loop. But it can also be offered to the user interface as a possibility to generate on-line hydrographs at a point selected by the user within the basin. The same algorithm is applied, and the only difference is that in the first case it operates on the results of the runoff generation during one time step of the computation loop and in the second case it uses the average runoff generation during a period of uniform rainfall. Other variants of the algorithm can also be conceived, such as the generation of a unit hydrograph, which characterizes basin state by evaluating basin response to uniform rainfall of unit intensity and a given duration. The methods of the *Gauge* object are reviewed hereafter. Two groups are of interest: administrative functions and behavior functions.

Administrative functions

Administrative functions take care of the creation and destruction of the data structures related to the *Gauge* object. The main initialization function is *init_routing()*. *init_routing()* performs the necessary operations to initialize an instance of the *Gauge* data structure for a point in the basin identified by its coordinates. It uses the *get_area()* function of the *Basin* object to obtain the contributing area to the point, and allocates memory for the routing map. It then calls the function *init_route_map()*,

which initializes the data structures that store the neccessary information about the pixels located upstream the outlet. At the end of the initialization, *Gauge* contains information about which pixels are located upstream the outlet and their corresponding distances of travel along hillslope and channel. Other initialization functions are *init_results()* and *init_gau_trace()*, which allocate memory for hydrograph storage and trace infromation respectively. Destruction functions free the memory allocated for the corresponding objects when they are no longer needed.

Behavior functions

The *Gauge* object controls the routing operations, and therefore, its behavior functions implement several versions of the distributed convolution described in Chapter 2, adapted to different circumstances. The main behavior function is *route_hydrograph()*, which obtains the incremental basin response for a computation time step. It first obtains the travel velocities in hillslope and channel as a function of the discharge at the basin final outlet for that time step, using Equations (2.46) and (2.47). Then, *route_hydrograph()* loops over the pixels upstream the gauge and estimates the time of travel for them, according to expression (2.40). It finally stores the contribution of the pixel to basin response, delayed by the time of travel. *unit_hydrograph()* and *route_pix()* implement variants of the algorithm for the cases in which the objective is to obtain the response of the basin to uniform rainfall or the hydrograph generated by one single pixel. There are also other methods of the *Gauge* object that implement convenient functions, such as *set_distance()* or *set_veocity()*.

The *GaugeTrace* object offers complementary functionalities, mostly for debugging and calibration purposes. If the trace is activated, the function *update_trace()* is called during the loop. The trace provides a decomposition of the origin of basin response, differentiating runoff according to two criteria. The first one refers to the runoff-generation mechanism, and discriminates between infiltration-excess runoff and return flow . The second criterion refers to location within the basin, and discriminates between runoff generated in areas which are permanently saturated because the water table is initially at the surface and runoff generated in areas which become saturated during the storm as a consequence of the dynamics of the model. The activation of the trace increases the computation time, but, combined with the basin trace, provides excellent information about the dynamics of runoff generation.

5.4 Implementation of simulation management

The *Simulator* object is in charge of controlling the simulations performed with the model. The *Simulator* object has two main functions: (1) it interacts with the operating system to get information from the hydrologic database and to store and display model results, and (2) it controls the evolution of model operation, deciding when new data should be read or when results should be stored. Model operation is based on three functional units: computation loop, rainfall loop, forecasting loop.

5.4.1 Input-output operations

Most of the input/output information is stored in the *IOData* data structure. All input/output operations are carried out by the *Simulator* object. It handles the different channels of communication, depending on the mode of operation and the input source selected. Input operations are of two main classes: those carried out during model initialization and those carried out periodically during model operation.

Initial input operations are encoded in the functions *read_env_var()* and *read_gen_data()*. The function *read_env_var()* reads from the UNIX environment the settings corresponding to the simulation, mainly paths and file keywords for the different types of information and values of model lumped parameters. The function *read_gen_data()* reads the files corresponding to the static distributed variables. *read_gen_data()* is also in charge of building the basin map, according to the header description of the raster format. It is assumed that all distributed variables correspond to the same basic *Raster* structure, and therefore only one map is needed to relate raster and vector descriptions of the variables.

Periodic input operations are those repeated cyclically during the operational loop. Two types of periodic input operations are considered: rainfall input and basin state. Rainfall input refers to measured rain and to the optional forecasted rain. The functions *get_next_mrain()* and *get_next_frain()* obtain the time tag of the next rain file from the active input channel at that time (either the standard input or a collection of files in a directory). The functions *read_meas_rain()* and *read_fore_rain()* compose the adequate file names using the path and file naming information stored in *IOdata* and load the contents of the raster files into

the corresponding array of the basin structure. The functions *set_input_list()*, *order()*, *comp_meas()* and *comp_fore()* are auxiliary functions to scan and sort the directory in the case of input from existing files.

Basin state is defined by the three state variables: wetting front position, top front position and moisture content. When forecasted rainfall is used or when the basin is initialized at an initial state different from zero, basin state is read using *read_initial state()* (first time) or *read_inter_state()* (periodically). These functions construct the adequate file names combining path, root, time tag and extension and call the auxiliary functions *read_last_nf()*, *read_last_nt()*, *read_last_dMt()* and *read_last_hyd()*.

Output operations are very similar to periodic input operations, since the same variables, path, and file names are involved. The functions *write_inter_state()*, *write_inter_hyd()* and *write_fore_hyd()* are used to write intermediate and final results (basin state variables and hydrographs). Other functions involving output operations are those writing trace results. They are invoked periodically or at the end of the simulation for the active traces.

5.4.2 Model operation

The second group of functions of the *Simulator* object are those related to the control of model operation, which involves computational, rainfall and forecasting loops. The computational loop is defined in the function *hydrograph_loop()*, and corresponds to one single computation time step. The rainfall loop corresponds to computations during a period

of constant rainfall, and usually involves several computational loops. The forecasting loop corresponds to the operational cycle 'measured rainfall-forecasted rainfall', and involves one or more rainfall loops. The rainfall and forecasting loops are implemented in the function *sim_loop()*, which follows the simple scheme 'get new rainfall - compute' for the simulation mode and 'get measured rainfall - compute - get forecasted rainfall - compute' for the forecasting mode. Each 'compute' stage involves calling the rainfall loop.

The computational loop corresponds to the inference of basin evolution during a computation time step. The simulator object only coordinates the functionalities of the *Basin* and the *Gauge* objects to compute basin evolution. The sequence of operations if the following: first, the lateral flow resulting from non-uniform distribution is computed, using the *Basin* function *para_flow_loop()*. Then, the infiltration model loop is called (function *hill_loop()*). When pixel evolution is completed, the routing functions (*hydrograph_loop()*) are called for the active gauges in the list.

The function *sim_loop()* provides the functional units of the rainfall and the forecasting loops. The basic task of *Simulator* in model operation is time control. The time step of the computational loop is fixed, but the duration of the rainfall cycles is not known a priori. Therefore, the *Simulator* object must decide the number of repetitions of the computational loop that correspond to each rainfall file received. Obviously, there is a trade-off between operational flexibility and computational efficiency, since a small time step reduces the error in rainfall validity time but increases computation time. In any case, model

operation is greatly simplified if rainfall files are guaranteed to be equally spaced in time.

The treatment of measured rain and forecasted rain is different. When a new measured rain file is received, it is assumed to correspond to the time interval transcurred since the last measured rain file, and the number of computation loops is decided accordingly. The duration of rainfall forecasts is known a priori, and therefore its operation is simpler. During model operation, *sim_loop()* calls the necessary I/O functions to obtain and store basin state, according to the active operation mode. In simulation mode, no state files are written unless explicitly stated by the user. In forecasting mode, either on-line or off-line, basin state is written at the end of the measured rainfall loop, and read again at the beginning of the new operational cycle.

CHAPTER 6

Software Design of the User Interface

The interface between RIBS and model users is discussed in this chapter. We first describe the software environment in which the interface is developed and the overall concept of user-model interaction. RIBS provides two user interfaces: a model-driven interface which advances synchronously with the model and a user-driven interface which is intended for off-line consultation of the database. A presentation of the software components of both interfaces is given in this chapter. Appendix 2, RIBS user manual, gives an overview of user interface functionalities, and a detailed description of software implementation is given in Appendix 5.

6.1 Design overview

This section presents an overview of the different components of the graphic software developed for user interaction. The system is intended to run on an engineering workstation over a computer network. Graphics in the RIBS system are hardware independent and network transparent because all graphic operations are conducted through the X Window protocol. The possibilities offered by the X Window programming environment are discussed first, and a description of the motivation and the design goals of the user interface is offered in Section 6.1.2.

6.1.1 The X Windows programming environment

The X Window system (Scheifer and Gettys, 1991) is a software environment for engineering workstations. It provides a client-server model where client applications can use a local program (the *X Server*) to handle input-output functions. The X Window system is network transparent and device independent. Network transparency implies that applications can be run on whatever host cpu is most convenient. The client programs and the *X Server* can reside in the same machine or reside in different machines and communicate through a computer network. Server and client programs communicate using a network protocol of messages that specifies graphic operations. The existence of a standard protocol confers device independence, since client programs do not need to know anything about the hardware they are using to interact with the user. The hardware is directly controlled by the *X Server*, which translates requests from the clients into graphic operations on the screen and sends user input back to the client programs.

From the standpoint of application programming, the existence of the X protocol means that X applications are portable. An application can be run remotely on a host machine over the network and display graphic results in different types of hardware. The application is therefore simultaneously available to all terminals connected through the network, and does not require additional software to be run on newly developed hardware. Conversely, from the user standpoint, an X workstation appears as connected to a number of different hosts at the same time, to

the point that applications running locally cannot be distinguished from applications running on remote hosts.

Application programmers do not use the X Window protocol directly. They usually gain access to the protocol through a number of libraries whose purpose is to let the application interface with the protocol and hide the complexity of the window system. The C language function library known as *Xlib* (O'Reilly, 1989) is used in this work. *Xlib* provides a model for raster graphic specification and a basic set of primitive window operations.

The X Window system also provides other libraries to develop user interfaces, called "X toolkits". A toolkit is a framework to combine user interface components, called "widgets", to produce complete user interface operations. The standard X Window toolkit intrinsics library (*Xt* library) is used in this thesis, together with the *Athena* widget set. The X toolkit intrinsics library offers general purpose user-interface functions and the widget set offers a number of graphical objects, such as simulated buttons, which are usually found in user interface applications. The widgets themselves are not a part of the X Window system. They are provided by other software vendors. The *Athena* widget set was selected because of its availability as part of the standard X Window instalation. However, only very simple widgets are used in the RIBS user interface, and therefore the adaptation of the code to more advanced widget sets, such as the *Motif* or the *Open Look*, is straightforward.

RIBS, as any other X application, also uses the functionalities provided by an specific program, called the "window manager", to handle user interaction. The window manager runs locally on the display workstation, and is in charge of interfacing between the *X Server* and the

client applications. The window manager lets the user control interactively certain parameters of the windows created by the applications, such as position, size or stacking order in the screen. It is up to client applications to provide the neccesary software to react to user actions on the windows. For instance, the window manager lets the user change the size of a window, but it does not update the contents of the window according to the new size. It simply sends a message to the client application notifying the new window parameters.

The RIBS user interface is therefore made up of four layers which interact with the user though a window manager. The lowest layer is the interface with the X Window protocol, *Xlib*. The next layer is the *Xt* toolkit intrinsic library, which, as *Xlib*, is part of standard X. The third layer consists of the widgets available in the *Athena* widget set, and the top layer is the RIBS application code, presented in this chapter.

6.1.2 The concept of RIBS user interface

The user interface is intended to coordinate interaction between the user and the model. The high-level design of the user interface answers two basic questions: (1) what information should be made available and (2) when should it be presented. The low-level design focuses on how those goals are achieved. The high-level and low-level design concepts applied in RIBS are discussed in this section.

High-level design

Two strategies are possible to specify the high-level design: model control and user control. Under a model-controlled strategy, the program makes automatic decisions about the presentation of results, without any feedback from the user. Model-controlled strategies are widely used in real-time systems, where models present results as they become available from the computations. Under a user-controlled strategy, it is the user who makes the decisions about which model results should be presented and how, and therefore the model must establish some interactive channel of communication to get user input. User-controlled strategies are mostly used in off-line contexts, to access data previously generated by other processes, such as in a database management system. The user-controlled strategy is obviously more versatile, but it involves a more complex software environment and a greater use of computer resources. The model-controlled strategy has the advantage of being able to respond to model events which might otherwise pass unnoticed by the user, but it cannot be made interactive.

The design of RIBS user interface is based on a combination of both strategies, because none of them can satisfy the requirements in isolation. Model results are too abundant to be presented simultaneously on one single computer screen, and therefore a model-controlled strategy would have to alternate the presentation of different variables, making it difficult for the user to follow the presentation. On the other hand, a user-controlled strategy is unaware of the state of model evolution, and the model does not have a mechanism to notify the user of important events, such as the completion of a time step and the availablilty of new model

results. Two modes of user-computer interface are included in RIBS: a model driven interface and a user driven interface. The model driven interface is basically model-controlled, but it lets the user interact with the computer to specify the format of the presentation. The user driven interface is basically user-controlled and is intended for off-line result analysis, but is has an open channel of communication with the model and can also be used in real time.

Low-level design

Two features of the distributed basin simulator characterize the low-level design of the user interface: the abundance and diversity of the data involved and the need for real time operation. Both features have interconnected effects. The model should run in real time, producing a dynamically changing image of basin state which should be offered to the user. Since model results are so abundant, they cannot be presented simultaneously to the user. On the other hand, the model evolves slowly, and the presentation of final output alone (model hydrographs exclusively) may be too simplistic. The typical time scale of changes in basin situation is of the order of hours. There is enough time for the user to assimilate and understand a significant portion of the results produced by the model as it advances in real time. The consequence is that a large volume of results are produced by the model and the user is potentially interested in all of them, but only a fraction will actually be analized in real time. The criteria to select relevant results change over time, as the model helps the user understand the complex situation in the basin. Different areas of the basin and different variables are of interest,

according to the situation. The user interface should offer optional access to very detailed representations of model results.

The strategy adopted to deal with abundant information is to distribute the presentation of results in several independent windows. On an X Window system workstation, the windows can be regarded as overlapping sheets of paper stacked on the screen. The window manager allows the user to manipulate each window individually, changing its stacking order, resizing it or iconifying it. The window manager also lets the user move windows around the screen. If the underlying application programs have capabilities to react to the different events generated by the window manager (exposure, resizing, iconization), as far as the user is concerned, the computer screen is just like his or her working desk top. Each window is like a sheet of paper that can be moved, stacked, put away and recovered. The design goal for the RIBS user interfaces is to create a working environment similar to the one just described to consult model results.

For the model-driven interface, results are presented on-line, as they are being generated by the numerical processes. Software is developed to present all types of results, but the user can select which results should be available for a particular run, in order to control the computational load. Once the model is running, windows with different variables are automatically updated as the model advances. The user can resize, iconify or magnify windows according to the situation. Zooming capabilities let the user focus on specific areas of the basin or analyze portions of the hydrographs with great detail.

The goal of the user-driven interface is to offer full access to the hydrologic database. The interface is conceived as an off-line analysis tool

which interactively offers some of the modeling capabilities of RIBS. RIBS stores intermediate basin states in the database. The user-driven interface is basically a program that retrieves the representation of basin state held by the model at any given time and operates on it to present specific information at the user's request. It can generate hydrographs at any point within the basin, display the time evolution of model variables for any grid element, or represent the spatial distribution of basic or derived variables. The user-driven interface provides access to a very large volume of model results without imposing the computational burden of generating those results on the model as it works in real time. These results about specific aspects are only computed when the user actually requests them.

The second problem of the low-level design of the interface is the need to operate in real time, which means that the model must be run concurrently with the interface. If result presentation is to be interactive, the process taking care of it must be waiting for user input, ready to react to user requests. The model, however, must follow the operational loop, and model evolution is driven by the arrival of rainfall information, not by user actions. When new rainfall arrives, the model must use its computational resources to process the information, and defining a cycle to attend periodically to user requests is computationally too expensive. From the standpoint of the user interface, the possibility to react promptly to *XEvents* means that the application must be running a loop waiting for events to arrive in order to process them. Therefore, the interactive user-interface must constitute a different process from the model itself.

The solution adopted is to have the model run in the background and let it present results using independent processes. That requires the

definition of isolated modules for result presentation and channels of communication between the model and the interface modules to guarantee prompt update of model results. The general manager starts the model and the user interface modules and controls communications between them. User interface modules can also start new modules.

In both interfaces, each module appears as an independent window with an identifying name. The interface as a whole is a group of processes which appears as a hierarchy of windows. The operating system takes care of the distribution of computational resources among all running modules and the window manager lets the user configure the global appearance of the screen. Windows containing information not required at a given time can be put away by iconifying them. Iconified windows display an icon resembling the application, which can be used to magnify them again. Resizing and zooming operations can also be performed in all graphic displays, and most user interaction is mouse-driven. The user can therefore customize the access to model results, choosing the variables that should be included in the presentation for every run and selecting which variables should be displayed on the screen at any given time.

6.2 The model-driven user interface

This section presents an overview of the different software components of the model-driven user interface. The general structure is summarized in Figure 6.1. The interface is composed of three basic modules: the pixel viewer, the hydrograph viewer and the raster viewer.

Each module is composed of a menu bar for user interaction and a graphic object. Graphic objects are derived from the *Image* class, which contains the core specifications for the RIBS graphic obects. Three subclasses of the *Image* class (one for each module) are defined: *PixImage*, *HydImage* and *RasImage*. The graphic objects are based on lower level objects for graphic operations and on the X Windows libraries. The low-level objects developed to deal with graphic operations are presented first. Then, the specifications for the *Image* class and derived subclasses are presented. The executable modules and their management by the interface are presented in the last subsection.

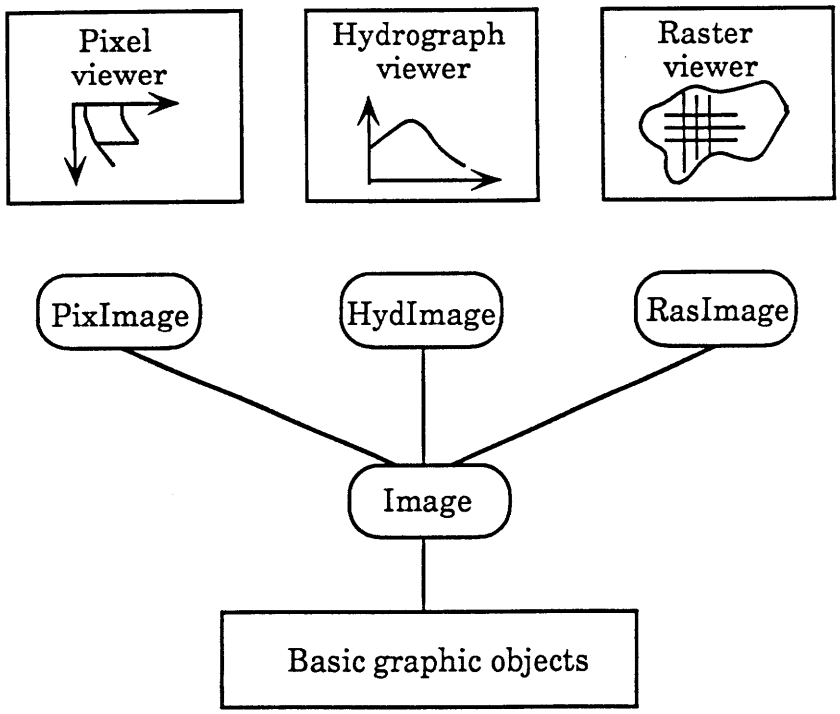


***Figure 6.1**: High-level components of the model-driven user interface.*

6.2.1 Basic objects

In addition to the *Xlib* and *Xt* libraries and the *Athena* widget set, the user interface is based on three basic graphic objects: *GrContext*, *GrReferenceSystem*, and *Graph*. *GrContext* deals with the graphic context as defined in the X model: colors, fonts, stipples, etc. *GrReferenceSystem* deals with the screen coordinate system, and *Graph* contains a basic graphic display based on the two previous objects. The object diagram of the basic graphic objects is shown in Figure 6.2.

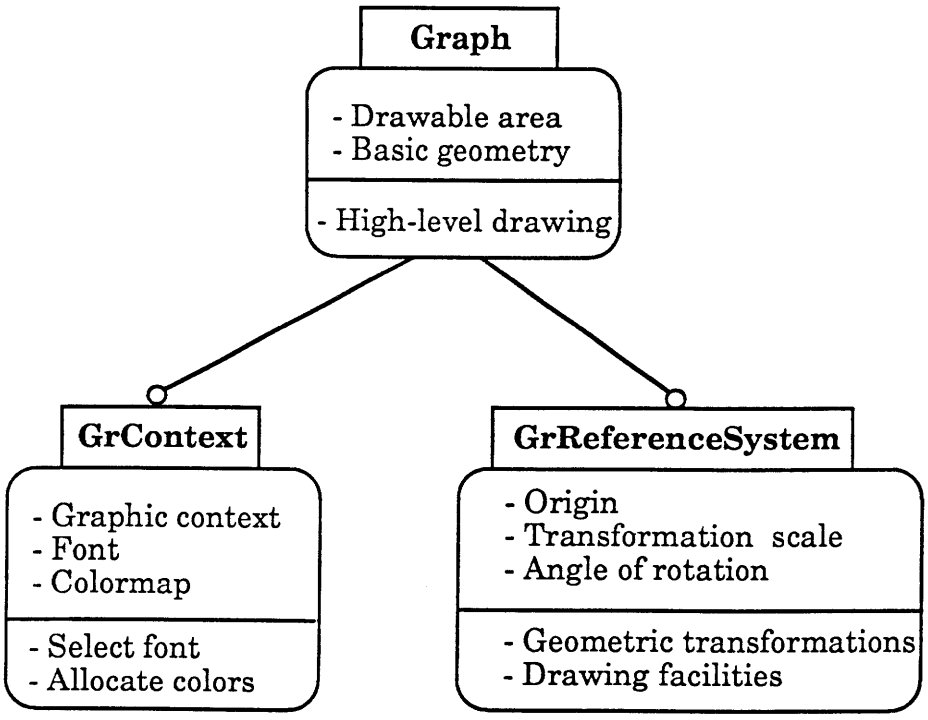


***Figure 6.2**: Structure of the basic graphic objects.*

GrContext

GrContext is a data structure which contains information about the graphic context used by a graphic object. The concept of graphic context is central in X Windows graphics. Graphic primitives requested by applications are processed by a graphic pipeline to convert graphic requests into pixels in the screen. The attribute values in the graphic context control several aspects about how the conversion is made. The *Xlib* library provides a corresponding graphic context data structure, *GC*, which specifies the attributes to use when drawing. The object *GrContext* encapsulates this data structure and allows the rest of the application to handle high-level graphic context descriptions. The *GrContext* object stores information about fonts and colors, and hides the black and white - color differentiation from the rest of the application.

GrContext contains pointers to the *Xlib* structures representing the graphic context (*GC*), font (*XFontStruct*) and color description (*Colormap*), and an array of *XPixmaps* to store shading patters for a grayscale. It also contains information about font geometry, number of colors or gray intensities and *RGB* color descriptions. Functionalities of the *GrContext* structure allow the user to select the adequate font according to the current screen size, and allocate and select colors from a user-defined color palette. If the screen does not have color capabilities, the *GrContext* object reserves a gray scale with different stipple patterns. Clients of the *GrContext* can therefore obtain information about front size or number of colors available and set graphic parameters according to simple high-level descriptions.

GrReferenceSystem

The function of the *GrReferenceSystem* object is to enable drawing using application-defined coordinate systems. Since all drawing functions in the *Xlib* library take arguments referred to the standard screen coordinate system, their use in high-level graphic applications is cumbersome. The *GrReferenceSystem* object maintains the equivalence between a user-defined coordinate system and the current screen coordinate system, which is a function of screen size, and provides drawing functionalities equivalent to those offered by the *Xlib* library, but referenced to the local coordinate system.

GrReferenceSystem stores the description of the current coordinate system in terms of position of the origin, transformation scales in both axes and angle of rotation. *GrReferenceSystem* provides two types of functionalities: geometric transformations and drawing facilities. Geometric transformations enable the user to change the local coordinate system and offer two-way coordinate transformations. Drawing facilities offer point, line and area drawing in the local coordinate system. The user can also place text with different alignment parameters.

Graph

Graph is a higher-level object that uses the functionalities of the *GrContext* and the *GrReferenceSystem* objects to perform generic drawing operations. *Graph* holds a *XPixmap* pointer to represent the area where drawing operations are to be performed. Since the pixmap can be either a window on the screen or a memory area, *Graph* can be used to

store images in order to display them in sequence later. *Graph* also maintains a graphic context, a reference system and basic geometric information. According to how many drawing areas are simultaneously maintained in the same widget, an application can create as many instances of the *Graph* object as required, and draw on them using different contexts or reference systems. *Graph* also offers methods to draw simple graphic figures, such as coordinate axes, grids or labels in plots, based on high-level descriptions and parameters.

6.2.2 RIBS graphic objects

The class *Image* contains the basic definition of RIBS objects requiring graphic representation. From the standpoint of object-oriented design, an image is the graphic representation of an object, and should remain attached to it as another attribute. In practice, however, it is more efficient to define graphic objects as independent entities in the hierarchy, in order to specify collective graphic operations for all of them. Graphic objects in RIBS are therefore defined independently from their corresponding physical objects. The essential structure and behavior of graphic objects are defined in the *Image* class. Graphic objects corresponding to specific physical objects are represented as subclasses of the *Image* class, and inherit the basic definition.

Only three RIBS objects are represented graphically in the user interface: *Raster*, *Hydrograph* and *Pixel*. Two of them, *Raster* and *Hydrograph*, are interesting because they are the objects stored in the database, and the third is the core object of the DBS model and of the application. RIBS graphic objects represent the connection between the

original data in the objects and their representation in the application window. The three graphic objects are *RasImage*, *HydImage* and *PixImage*, which are subclasses of the *Image* class. Figure 6.3 illustrates the structure of the *Image* class and its subclasses.

The Image class

In RIBS, an *Image* is a group of *Graph* objects which are collectively represented in the same widget and can react to basic graphic events. All graphic objects have the same graphic capabilities, inherited from the

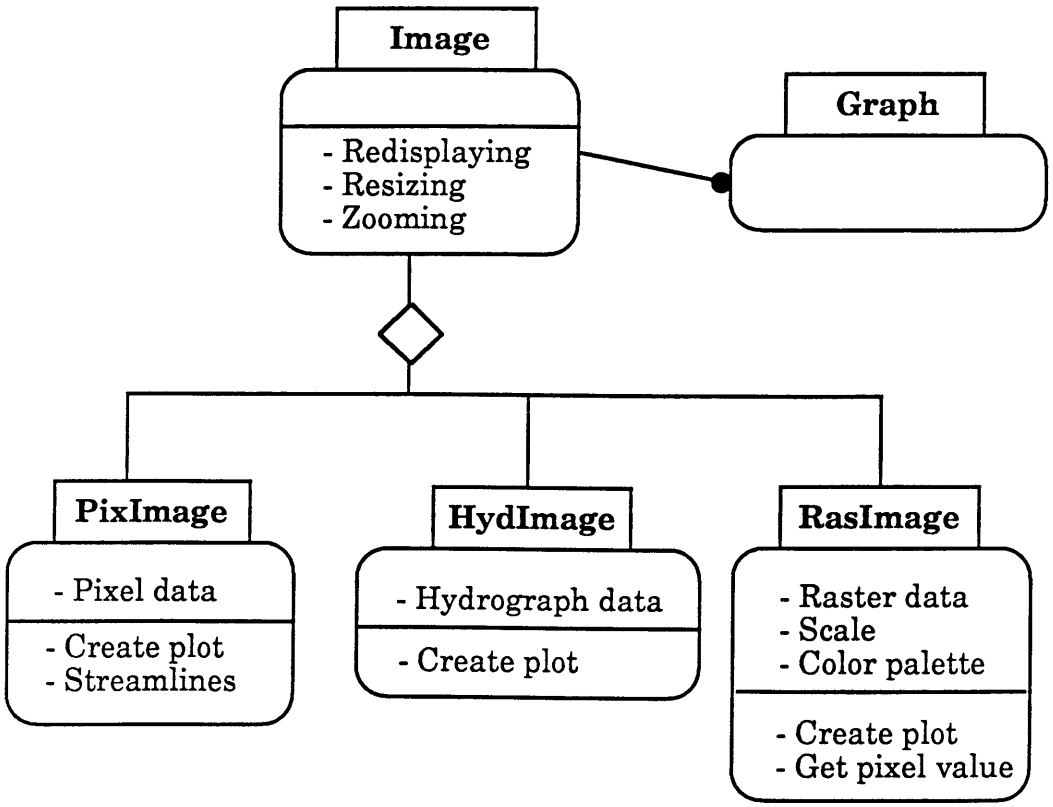


***Figure 6.3**: Structure of the Image class.*

parent class, *Image*. Graphic objects are represented on a window which is resizeable by the user. They must therefore respond to two basic graphic events: *expose* and *resize*.

An *expose* event is sent by the *X Server* to the application whenever the window exposure changes. That announces that part or all the window needs redrawing because some other window has just stopped covering it. Although certain window managers store backups of hidden areas of windows and can take care of redrawing whenever appropriate, this feature is not guaranteed in the *Xlib* standard, and client applications must process *expose* events to prevent that other applications whose windows pop up or are relocated leave holes in the window of interest when they are removed.

Objects of the *Image* class handle the *expose* event in the *redisplay()* function. They apply the standard technique of drawing into a backup graphic buffer (an instance of the *Graph* object), instead of drawing directly on the screen. The image is therefore stored in an auxiliary memory area, and whenever an *expose* event is generated for the window, the backup image is copied on the window again. Since the *XCopyArea()* function is significantly faster than the *resize()* functions that generate the drawing, the extra memory requirement is justified by the fact that the window can be redrawn as many times as neccessary with almost no processing time.

A *resize* event is generated whenever the window manager changes the external size of a window, either due to a user request or to some resource modification in the programs. Objects in the *Image* class react to *resize* events by redrawing the image, adjusting it to the new window size. Every subclass of the *Image* class implements a different version of the

resize() method, since it involves displaying a diferent data structure on the screen. The fraction of *resize()* shared by all subclasses of *Image* involves querying the *X Server* for the new values of the widget size resources, adjusting the reference system to the new screen coordinates and selecting the font size adequate to the new window size. Once the new settings heve been defined, the specific function to draw the image is called. Since objects in the *Image* class usually draw in the pixmap of the *Graph* object associated to them, they must also copy the image on the screen by calling the method *redisplay()*.

In addition to the mentioned facitilies to react to window managing, the *Image* class also offers interactive zooming capabilities, implemented in the functions *do_zoom()* and *do_unzoom()*. The zoomming functions follow a two-step process. In the first step, the new focus area must be selected from the picture on the window. *do_zoom()* uses interactive rubber-banding techniques: the user clicks the mouse on the picture to select in one of the corners of the new image. Then, as the mouse slides within the picture with the button pressed, a rectangle is drawn on the screen showing the current selected boundary for the new image. When the button is released, the selection is finished and the rectangle is erased. In the second step, the screen coordinates are transformed into the local coordinate system to obtain the new image boundaries. The reference system is then changed and the picture is drawn again with the new settings. The *do_unzoom()* function follows an inverse procedure, resetting the picture to its original boundaries.

PixImage

The *PixImage* object is the graphic representation of the RIBS Pixel object. The *PixImage* object implements the functional unit described in Section 4.5.2, and generates the picture shown in Figure 4.6. It also may optionally pop up a window and display a cross section along the line of maximum slope, displaying flow streamlines. The variables stored in the *PixImage* object are those required to draw the moisture profile of the soil column: front and water table positions, moisture content, terrain slope, Brooks-Corey parameters and initial recharge rate. It also stores the pixel denomination and *Graph* objects to store the drawings of the moisture profile and streamlines. *PixImage* offers the basic functionalities included in the *Image* class, plus the function *create_pixplot()*, which actually draws the picture shown in Figure 4.6. *create_pixplot()* is used by *resize()* to generate the drawing after the new window size has been specified.

HydImage

The *HydImage* object displays hydrographs stored in the database. It generates the graph shown in Figure 4.7. The data structure of the object *HydImage* includes the definition of the hydrographs and hyetographs: number and denomination of the hydrographs, number of data points in every one, and actual values. It also contains the *Graph* object that is used to store the drawing. In addition to the methods of its parent class, the object *HydImage* has the method *create_plot()* to generate the actual picture of the hydrograph display.

RasImage

The object *RasImage* represents the image of a *Raster* object in the application window, as shown in Figure 4.8. Although the *RasImage* object is also a subclass of the *Image* class, it has a more complex structure than the other two subclasses. The picture consists of four areas, each one associated to a different widget: The title widget, the raster widget, the scale widget and the label widget. All but the label widget are drawing areas, and are therefore represented by *Graph* objects. The *RasImage* data structure also holds a pointer to the *RasterHeader* and an array with the actual values. Finally, the *RasImage* object also contains information about the scale and the color codes used to represent different values of the distributed variable.

The *RasImage* object represents the values of the distributed variable following a color code. The color code is decided according to a linear scale, which divides the total range in a number of intervals. Pixels whose value is within the limits of an interval are drawn in the color associated with the interval. Also, special colors are reserved for pixels that underflow or overflow the current scale. Both the scale (upper and lower limits and number of intervals) and the color code (color palette) can be specified by the user. In order to provide a reference, a scale is shown on the left of the plot. The scale shows the colors which are actually represented in the plot, together with the maximum and minimum values of the variable. The display is completed with a title bar that includes generic information (variable designation and coordinate system, size and location of the plot) and a label on the upper left which is

used to display specific information (such as the actual value of a selected pixel).

RasImage inherits the functionalities of its parent class to handle the graphic display. The function *create_image()* is used by *resize()* to draw the scale and the raster display on the auxiliary *Graph* objects. It also implements specific functions to define and change the color palette (*set_palette()*) or the scale (*set_scale()*), which can be activated either at the beginning, to define initial settings, or at run-time, to change settings for a given picture. Another functionality provided by *RasImage* is the display of values of individual pixels. The user selects a single pixel in the display by clicking the mouse, and the variable value for that pixel appears in the label screen on the upper left corner of the window.

6.2.3 Executable modules

Three executable modules are built using the functionalities of the *Image* class: the pixel viewer (*x_pixgraf*), the hydrograph viewer (*x_hydgraf*) and the raster viewer (*x_rasgraf*). Although specific software was written for each module, the executable modules for the object viewers were all built using the same scheme. A high-level *main()* function creates the necessary objects and widget hierarchy and uses the *Image* class methods as callback functions for the different events. Additional functions are also written to create the graphic context and to initialize the *Image* objects reading the files from the database.

The layout of the object viewer modules is shown in Figure 6.4. The main application window is a pane widget divided in two areas: an upper command area containing a menu bar and a main graphic area

containing the display of the *Image* object. The command area is a box widget that contains several command button widgets. Each button provides access to one functionality and is linked to a method of the *Image* class or the corresponding subclass. Buttons are activated by clicking the mouse within them.

Four basic command buttons are available in all object viewers: *File*, *Zoom*, *Unzoom* and *Quit*. Applications can also define new buttons to provide access to other functionalities. The *Quit* button, which terminates the application, is optional, since in some cases the viewer must be controlled by its parent process and cannot be terminated by the user. The *Zoom* and *Unzoom* buttons provide access to the zooming capabilities described in the previous section. The *File* button allows the user to change the file currrently being displayed. A window pops up with the

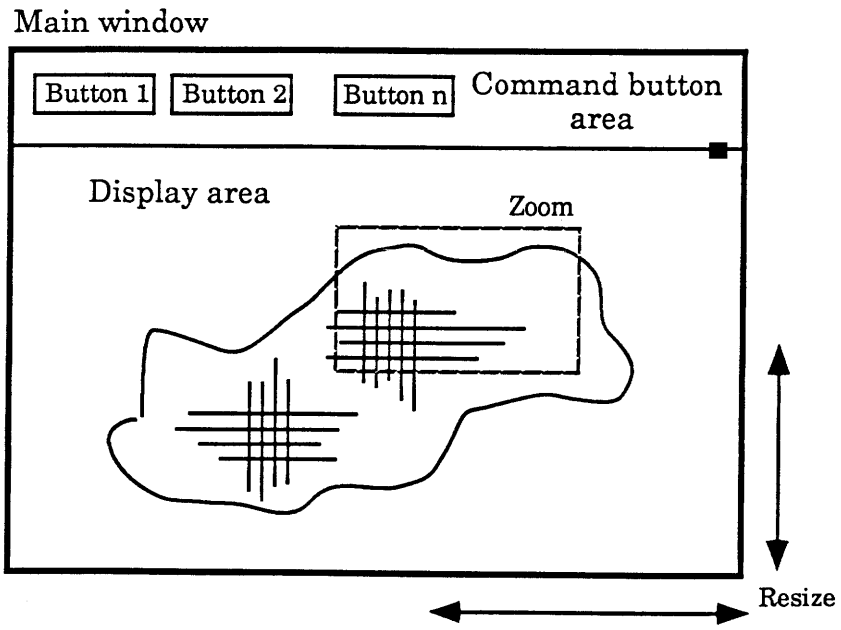


***Figure 6.4**: Basic window layout for the object viewers.*

contents of the current directory and the user can interactively select a new file to display. The application must have a callback function to react to this event, reading the contents of the new file and calling the corresponding *resize()* function to display it on the screen. In addition to this menu-driven file change, the applications also accept new file names from the standard input. The user (or the parent process if the viewer was started from a pipe) can type a file name and the application reacts displaying the new file on the window.

The graphic area is another box widget which contains the widgets of the corresponding subclass of the *Image* class. The graphic area is created with a default size, but the user can resize it using the window manager. The application must take care of assigning the adequate event handlers to the *expose* (function *redisplay()*) and *resize* (function *resize()*) events.

From the software engineering perspective, individual applications must take care of two main activities: initialization of the widget hierarchy and definition of the environment. The widget hierarchy is the structure of widgets maintained by the application. Widgets are created and initialized in the *main()* function of the corresponding application. The user can define widget resources, such as window names, colors or geometries, either in the *.Xdefaults* file or in the command line. If no user-specified parameters are declared, widgets are initialized using fallback resources that are hardcoded in the program.

Applications must also provide functions to initialize the graphic environment and access the database. The initialization of the graphic environment is carried out in the function *initialize_data()*. If a color display is available, *initialize_data()* allocates color cells in the active color

map. If the color map has free cells (not used by other aplications), the colors are initialized with the exact user-defined *RGB* values. If no more color cells are available, *initialize_data()* selects for every color the cell that more closely matches the user definition. If the available display is in black and white, a gray scale of shadings is loaded instead. The initailization function also allocates memory for the *GrContext* and *GrReferenceSystem* objects and loads the font structure.

Database access is provided by the function *read_file()*, which is used by the callback functions that react to 'new file name' events, comming either from the standard input or from the *File* button. *read_file()* reads the corresponding file from the database and stores the data into the *Image* object.

6.2.4 The model-driven interface

The synchronous user-interface is part of the general manager, and it is built upon two object viewer modules: *x_rasgraf* and *x_hydgraf*. Several processes corresponding to these modules are running simultaneously. Each module presents a variable or hydrograph as a graphic object, allowing for user interaction. Processes have open channels of communication with the numerical models, and are ready to change the display according to model evolution. Object viewer modules accept new file names from the standard input as a part of the *Xt* main loop, and therefore their parent processes can send them messages through standard input every time a new file needs to be displayed. Each variable appears on the screen in an individual window, which is ready to accept user input to resize, zoom or perform specific functions.

Two types of information are presented in the interface: rainfall information and basin state information. The rainfall information includes raster displays of measured and forecasted rainfall and a hydrograph display of hyetographs of rainfall registered at selected locations in the basin. The basin state information inlcudes raster displays of the three basin state variables: top front, wetting front and moisture content, raster display of runoff generation and hydrographs at points of interest in the basin.

The user decides which of those possible displays should be active for a particular run. Color palettes and scales can also be specified by the user. Each graphic object appears as an independent window with an identifying name. If windows are iconified they display a icon resembling the application. As the model progresses, new files are being generated. The displays are periodically updated with the latest information, but they also let the user select previous files using the *File* option. The user can therefore customize the access to model results, choosing the variables that should be included in the presentation for every run and selecting which variables should be displayed on the screen at any given time.

6.3 The user-driven interface

The idea of the user-driven interface is derived from the concept of object-oriented design. RIBS is basically an object-oriented modeling environment that obtains the state of physical objects as it evolves in time. Since the time evolution of model objects cannot be maintained in memory, objects defining the state of the system are stored in a database.

The user-driven interface can retrieve those objects from the database and apply the modeling capabilities of the object-oriented environment to them. Furthermore, it also offers graphic interactive facilities to communicate with the user.

The objective of the user-driven interface is twofold. First, it provides an integrated mechanism to access the entities in the hydrologic database: basin variables and hydrographs. That same objective could be attained by the individual modules of the model-driven interface presented in the previous section, but they only deal with the variables as individual data files. The user-driven interface deals with state variables in an integrated fashion, adding knowledge about time evolution and the meaning of the variables involved. It understands how data are organized in the database and guides the user during the consultation. The user-dirven interface knows about the time organization of RIBS, and can interpret the semantic content of the different data files. It knows which variables represent rainfall, runoff, front positions, etc, and can apply different scales or color codes to the different types of variables.

The second objective of the user-driven interface is to provide additional modeling capabilities for the RIBS environment. The distributed basin simulator has a modular structure, where the evaluation of basin evolution and the generation of basin response constitute independent processes. Once basin states have been obtained for a particular storm, the modeling environment offers a large number of possibilities with little additional effort. However, the automatic evaluation of all the variables and results that can be of potential interest to the user is neither essential nor efficient for a process intended to work in real time. Therefore, the on-line version of DBS focuses in basin

evolution and in obtaining a reduced number of basin hydrographs, leaving additional modeling capabilities for a complementary process. This second process, the user-driven interface, is directly controlled by the user, and only performs the computations that are considered relevant by the decision maker.

The user-driven interface should therefore work asynchronously with respect to the model. It should be guided by the user, not by model evolution. The objective is to build an environment where the user decides which previous basin state should be retrieved and what operations should be performed on it. The basic modules for graphic interaction are already available from the model-driven interface. The specific design for the user-driven interface should concentrate on enhancing the capabilities of the *Simulator* object to manage model inference and providing adequate interactive access within the context of the user interface. The basic object developed to perform that function is described in Section 6.3.1, and the executable module that implements the interface is presented in section 6.3.2.

6.3.1 The *Viewer* object

The *Viewer* object is the basic object of the asynchronous user interface. It consists basically of a *Simulator* object and an *Image* object, as shown in Figure 6.5. The *Viewer* object is just a manager of the functionalities provided by the other two objects, and does not store any other data structure. The *Viewer* uses the methods of the *Image* object to control interaction with the user and the methods of the *Simulator* object to carry out model computations. The *Simulator* in turn manages the

functionalities of the *Pixel*, *Basin* and *Gauge* objects. The user interface is therefore built as an interactive model interpreter, in which different aspects of model capabilities are offered to the user.

Functionalities added by the *Viewer* object are based on its capability to interact with the database to access past model results. The *Viewer* object offers the possibility to recover from the data base the model state corresponding to any past time. Model state is defined by the basin state variables (wetting front, top front and moisture content), and the forcing (the rainfall intensity and the runoff rate). Two methods of the *Simulator*, *read_initial_state()* and *read_initial_forcing()*, are used by the *Viewer* to obtain the state of the basin corresponding to a certain time tag. The request for a model state may come directly from the user (through menu

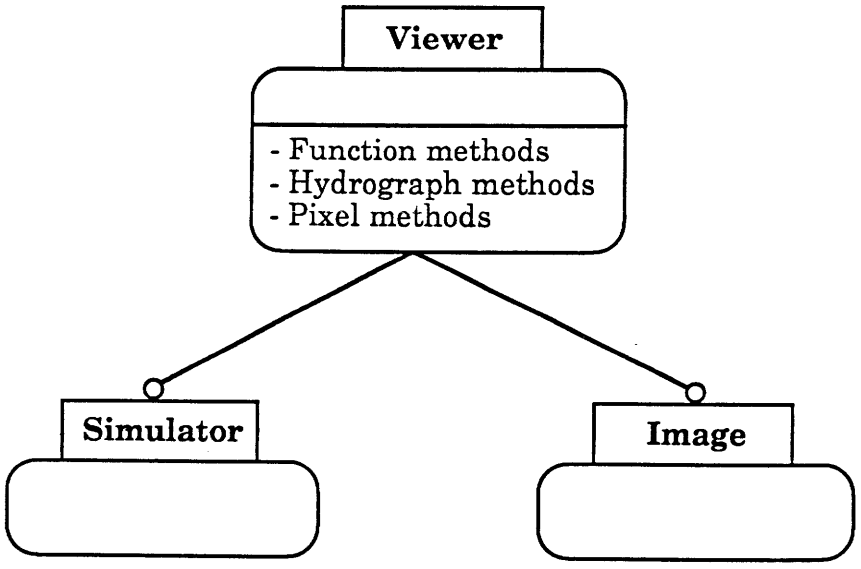


Figure 6.5: Structure of the Viewer object.

interaction), from an external parent process updating the *Viewer* (through the standard input), or as part of some *Viewer* method (through message passing). Once model state has been recovered, member functions of any of the physical objects can be applied to reconstruct model computations or to generate new results based on basin state.

The methods of the *Viewer* object belong to three basic groups: function methods, pixel methods and hydrograph methods. They correspond to the three main types of computations carried out by the viewer: function methods apply a function to several pixels in the basin at a given time, pixel methods apply a function to a single pixel at several times and hydrograph methods apply the convolution operation. Each group of methods is represented as a different option in the menu bar.

Function methods

Function methods are used to display virtual variables on the screen. Virtual variables are magnitudes which can be derived from basin state, as defined in the basin viewer: wetting and top front position, moisture content, runoff generation rate and rainfall rate. A complete list of the methods available in this group is given in Appendix 2. The list includes variables available in the current implementation of RIBS, but it can be easily modified to include other variables. The structure of all these methods is the same: the function that defines the virtual variable is applied to every pixel in the domain of application, and the resulting distributed variable is displyed on the window using the *Image* object.

The availability of virtual variable is extremely useful for model development and understanding. Most quantities used during model

inference can be represented as virtual variables. Model dynamics are formulated in terms of the state variables, and a wealth of information about the soil column can be obtained from their knowledge. Ultimately, any quantity defined for the one-dimensional model of infiltration can be expressed in terms of the three state variables and the external forcing. The only quantities which cannot be formulated using only local knowledge are those which refer to pixel interactions. Examples of derived magnitudes are surface infiltration capacity, front speeds, moisture storage in different areas, normal and parallel flow at any depth, total lateral flow, etc. The user can test and understand model behavior by visualizing the proper variables: the distribution of surface and subsurface runoff, the distribution of pixels in every state, runoff generated by pixels in a certain state, etc. The great advantage of using object-oriented programing is that the same code (object methods) used in model computation is also applied in user interface presentations. Therefore, every single step of model inference can be made available to the user for detailed analysis.

Pixel methods

Pixel methods consider one single pixel in isolation. The most simple method, *do_pixval()*, just offers the value for that pixel of the currently active virtual variable. This method uses the auxiliary function *get_pixel_coord()*, which lets the user select a pixel interactively with the mouse. As the user moves the mouse with the button pressed, the corresponding pixel is highlighted and the position of the pixel and the value of the variable are displayed in the label widget of the Image object.

The method *do_pixdis()* presents the complete state of a pixel. The user selects the pixel, and a report of pixel state is generated and presented through a *x_pixgraf* process.

The third pixel method, implemented in the function *do_pixrep()*, offers the time evolution of the currently active virtual variable for a pixel. The method involves recovering all basin states since the beginning of the simulation until the currently active time and applying the virtual function to the pixel. The time series generated is stored in a Results data structure and is displayed using a *x_hydgraf* process. The display includes also a report of the hyetograph in the pixel, both for ease of interpretation of the variable evolution (which will obviously be related to rainfall) and for completeness in the presentation. All displays of temporal evolutions are simultaneously maintained in the screen until the user explicitly terminates the process. The *do_pixrep()* method is a very powerful tool to compare the evolution of a variable for several pixels or to compare the evolution of several variables for the same pixel.

Hydrograph methods

The goal of hydrograph methods is to generate different types of hydrographs at specific locations selected by the user. Since basin state includes the average runoff generation rate for every rain step, all the viewer needs to do to generate hydrographs at any point in the basin is to apply the distributed convolution to the area located upstream the selected point. During model inference, DBS works at computation time steps and only writes basin state when new rainfall information arrives. Modules in the user interface must work at the resolution at which model results are

available, and that introduces a small error in hydrographs generated by the *Viewer* object compared with those generated by DBS. Nevertheless, the error is usually negligible, because the variability of basin state within a period of uniform rainfall is very small.

The function *do_hyd()* implements the basic hydrograph-generating operation. *do_hyd()* calls *get_pixel_coord()* to obtain the location of the outlet. Then it creates an instance of the Gauge object at that point calling the *Simulator* method *init_hyd_gauge()*. The function *hydrograph_loop()*, which is also a *Simulator* method, is applied to obtain the basin response at the gauge. The last step is to write the results and start a *x_hydgraf* process to present the results. Most of the computational burden of DBS is placed on the estimation of runoff generation, and the surface flow routing is a much faster process whose application is perfectly feasible in an interactive fashion. Although these custom-made hydrographs involve reading the runoff generation file several times, they are generated in a relatively short time and they are a very powerful tool for real-time decision making. Streamflow forecasts can be available at any point in the basin, and the user can concentrate the computational efforts where they are needed the most.

In addition to these local hydrographs, other variants of the convolution operation can also be applied to model results. The behavior of individual pixels can be tested by computing their contribution to the total hydrograph, as in the method *do_pixhyd()*. Also, if time of travel is eliminated from the convolution, the result is the time evolution of runoff generation in any portion of the basin, which can be compared with rainfall to obtain the global runoff-generating behavior of the subbasin. Lastly, a type of surface-runoff unit hydrograph can also be generated.

Given basin state at a certain time, the viewer applies the function *do_unit_hyd()* to evaluate the surface runoff response to a uniform rainfall of unit intensity during a time step. Since basin state changes over time, the result obtained is not a unit hydrograph in the traditional sense of the term, it is rather the unit response function of the basin corresponding to a certain duration of rainfall. In any case, these unit response functions are extremely useful to characterize global basin state at some time. They are available for any point within the basin, and they can be used either to obtain a quick and rough estimate of basin response to future rainfall or to analyze the dynamic character of the runoff generation potential of the basin during the storm.

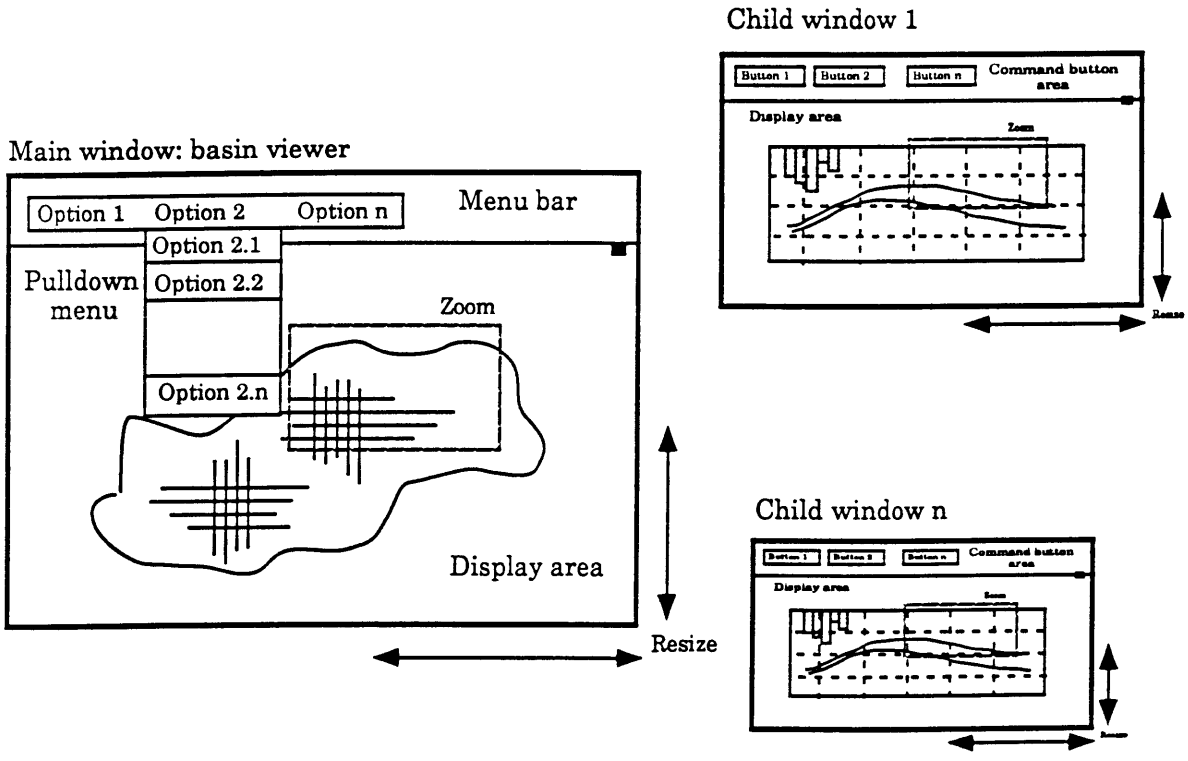


***Figure 6.6**: Basic window layout for the user-driven interface.*

6.3.2 The executable module

The program *x_bviewer* is the executable module that implements the asynchronous user interface. *x_bviewer* is simply an object-oriented graphic interface that connects the user with the functionalities offered by the *Viewer* object. The layout of the basin viewer main window is shown in Figure 6.6. The main window consists of a menu bar and a graphic area. The menu bar contains entries for several pulldown menu options. The graphic area contains the display of a *RasImage* object, as described in Section 6.2.2. The graphic area is used to display distributed basin variables and always shows the currently active virtual variable. Other results, such as hydrographs, temporal evolution of variables or pixel

File	Edit	Variable	Hydrograph	Pixel
Time Quit	Zoom Unzoom	Wetting front Top front Moisture content Runoff generation Rainfall Hydraulic conductivity Infiltration capacity Flow at front Upper deficit Upper saturation Lower deficit Total deficit Total saturation Distance to stream	Local hydrograph Pixel hydrograph Unit hydrograph	Pixel value Pixel display Variable report

***Figure 6.7**: Options in the pulldown menus of the basin viewer.*

states, are displayed using children windows which pop up as required. Children windows are actually entirely independent *x_hydgraf* or *x_pixgraf* processes, which are directly managed by the user.

Menu options of *x_bviewer* are detailed in Figure 6.7. The *File* menu option allows interaction with the database. Since the basin viewer is used to access results for a specific storm, the only *File* option available is a change in the time tag displayed. Time change can also be achieved by writing the new time tag in the standard input, and therefore the *x_bviewer* process can be controlled by a parent process in the same fashion as the other members of the user interface class. The *Edit* option allows user manipulation of the distributed basin display, such as changes in scale, color palettes or zooming operations. The other three menu options, *Variable*, *Pixel* and *Hydrograph*, give access to the specific methods of the *Viewer* object as classified in the previous section. *Variable* contains a list of the available virtual variables. *Pixel* contains options to consult variable values, display pixel states or present time evolution of the currently active virtual variable for individual pixels. The *Hydrograph* option allows the user to generate basin hydrographs or pixel hydrographs at a given location, obtain the current unit response function of the basin or compare rainfall with runoff generation.

The user recovers intermediate model states, and can then apply pieces of model inference to those states, testing model evolution. He can perform global consistency checks, by defining special variables, or he can just elaborate on primitive variables to obtain meaningful representations of basin state.

The user-driven interface is intended to run independently from the simulation model itself. It is basically a tool for detailed data analysis, and

therefore some of its capabilities are of little use in a real-time situation. However, it can also be run on-line with the simulation model, in which case it is automatically reset to the current time every time the model obtains a new set of results.

CHAPTER 7

Conclusion

7.1 Summary of results

This work presents a computer package called Real-time Interactive Basin Simulator. RIBS is a prototype computer implementation of a flood forecasting system. The system integrates a distributed rainfall-runoff model and a hydrologic database within a graphic computer environment which allows for real-time operational use. Although the RIBS system has a wider scope, the presentation concentrates on the distributed basin simulator and on the user interface.

The distributed basin simulator is a topography-based, rainfall-runoff model which can be used for real-time flood forecasting in midsize and large basins. Model use is specially attractive in connection with a meteorological radar and distributed rainfall forecasting methods. The model captures the main features of runoff generation processes in forested watersheds while keeping computational efficiency for real-time use. It accounts for the effects of slope, anisotropy and soil heterogeneity on subsurface flow and runoff production through a simple analytical formulation of infiltration processes at the hillslope scale. Basin-scale processes of subsurface and surface water transport are also adequately represented through discretized schemes on a rectangular grid.

Model conceptualization of the distributed basin simulator offers a variety of possibilities form the standpoint of its practical use in flood-forecasting schemes:

- It is based on the same spatial discretization as that used in Digital Elevation Maps, and therefore it can effectively incorporate very detailed topographical information.
- Basin state can be described with a reduced number of state variables. Model formulation in terms of state variables offers the possibility of storing intermediate basin states in the database for later retrieval. This feature can be used to present interactive reports of basin state and evolution or to split model evolution in several lines of action, considering, for instance, different future rainfall alternatives.
- The model can offer hydrographs at different locations in the basin simultaneously. This possibility is extended to be applicable in real time, with hydrographs being generated for points chosen interactively by the user.
- Model conceptualization is modular. The simulator is composed of a number of modular entities which are relatively self-contained. The evaluation of the runoff generation is entirely independent from its routing to the outlet, and both processes are separated from the management of the operational cycle. The same building modules can be used in different contexts.

The model was tested on an 840 km^2 catchment in the Arno basin (Italy), under the average "less-than-ideal" conditions that can be expected in a practical application, using a DEM of coarse spatial resolution, a previously available soil study, and a short record of ten

events. Rainfall was recorded on a sparse raingauge network and streamflow was measured only at one location. Five storms were used for calibration, saving the rest for an evaluation test. Calibration was based on the independence of the two main model parameters, which are responsible for surface and subsurface runoff respectively. Results of the evaluation process were encouraging, showing that, except for the definition of the initial state, the model can be calibrated to acceptable levels of performance with a limited data set.

The hydrologic model was implemented in *C* language using object-oriented design techniques. Rather than a single computer program, the product of the software development process is a library of simulation tools for distributed hydrology which can be used for a variety of purposes. The code implements the one-dimensional infiltration model applied at the subgrid scale, basin processes of runoff generation and flow routing, real-time simulation management and interaction with the hydrological database.

Model capabilities are accessed through a versatile user interface. Interface functions are used to present model results in real time and to access previous basin states stored in the database. The interface consists of interactive graphic programs working on a window environment, but interface functions are wider in scope, since they can be used by any other application to extract model results from the database.

The work presented here is significant because it integrates together a number of previously separate techniques and lays the foundation for possible future systems. Increasing data availability and computational power will probably lead to widespread use of distributed models in hydrology. The combination of hydrologic modeling techniques and

software development methods as presented here seems a promising alternative to apply physically-based concepts to flood forecasting.

7.2 Suggestions for future work

In the area of hydrologic modeling, the most important problem left unresolved in this work is the definition of basin state at the beginning of the storm. Event-based hydrologic modeling is basically an initial value problem. Moisture conditions at the beginning of the storm are crucial to estimate basin response, and future research should address this problem. Initial state in DBS is determined by the water table depth and the moisture content of the soil column above the water table. There are basically three ways of approaching the definition of an initial condition for DBS: (1) measure it, (2) model it and (3) estimate it in real time using streamflow measurements.

At the intended scale of operation of DBS, the measurement of soil moisture is clearly a problem of the future. Present remote-sensing techniques only provide an estimation of the moisture content of a very superficial soil layer, but future developments in the field may be able to measure greater depths, thus providing a reliable estimate of initial saturated areas at reasonable cost. An approximate estimate of the initial state can also be obtained applying an inter-storm soil moisture model. Model conceptualization in DBS is event-oriented, and does not include the capillary-controlled processes that govern soil moisture distribution on the long term. It is possible, however, to expand DBS, adding a different conceptualization to operate during inter-storm periods. A combination of

a vertically-averaged model of capillary soil moisture, similar to that of Blain and Milly (1991), and a water table position model, similar to that of Cabral et al. (1990), including moisture transfer between the saturated and the unsaturated zone, is probably the best scheme.

The third option to provide DBS with an initial state is to estimate it from streamflow measurements. This option assumes that the model is properly calibrated, and the initial condition is the only unknown of the problem. If that is true, a comparison of model predictions with actual measurements can be used as a basis for real-time model update, selecting the initial state which best reproduces the measurements. In order to keep the parsimony of the problem, the number or structure of possible initial states should be limited. The availability of several discharge measurements distributed throughout the basin can greatly benefit the filtering scheme.

The problem of subgrid variability is another hydrologic-modeling issue which is worth further research. The fact that the one-dimensional infiltration model is applied to grid cells in DBS is explained by the assumption that the conditions under which the infiltration model was derived are applicable to grid cells, and therefore all variables are considered uniform throughout the cell. In practice, the model should account for spatial heterogeneity, specially in large grid cells, where the assumptions of uniform slopes and front surfaces parallel to the terrain surface are weaker. If model variables represent spatial averages of random functions over the cell, the effect of their distributions on model equations should be investigated. Given the non-linearities in runoff generation, a stochastic version of the infiltration model could account for

subgrid heterogeneity and lead to different equations to characterize the external behavior of the cell.

From the point of view of practical use of the model, it is extremely interesting to test model behavior with actual radar-generated rainfall maps. The work initiated by Pessoa et al. (1992) with the preliminary version of the distributed rainfall-runoff model should be continued. The short-term goal would be to evaluate model performance with better data sets, but the generic final goal is the study of the influence of the spatial distribution of rainfall on basin response, which may have important implications in the fields of network design and real-time operational forecasting.

Given the limited scope of this work, the presentation of the RIBS package is incomplete from the standpoint of real-time operation. This work presents real-time simulation tools. In practice, at least two basic aspects should be added to model operation: distributed rainfall forecast and real-time update. Model operation can greatly benefit from the introduction of one or more radar-based rainfall-forecasting procedures. The model could be used to explore the implications of rainfall forecasts in terms of streamflows or water levels at different points in the basin. Also, if discharge measurements are available, the real-time operation system should account for them, updating model results and correcting model performance according to the measurements. As it was mentioned above, the single most important variable to modify in real time is the estimate of basin state at the beginning of the storm. However, if independent estimates of initial state are available, other model parameters, notably f and a_r, could also be tuned in real time using filtering schemes.

To complete a real-time decision-support system, the problem of general model inference should also be addressed. This work has concentrated on the definition of efficient simulation tools, but the automatic use of modeling in real time also requires the definition of inference strategies which use the simulation tools to approach the problem. The complexity of the changing situation during a flood will probably require the combination of numerical analysis with heuristic knowledge, leading to mixed architectures, as in other fields of engineering problem solving.

References

Abbot, M. B., J. C. Bathurst, J. A. Cunge, P. E. O'Connell and J. Rasmussen (1986), An introduction to the European Hydrological System - Systeme Hydrologique Europeen, "SHE". 1: History and philosophy of a physically-based, distributed modelling system, J. Hydrol., 87, 45-59.

Abbot, M. B., J. C. Bathurst, J. A. Cunge, P. E. O'Connell and J. Rasmussen (1986), An introduction to the European Hydrological System - Systeme Hydrologique Europeen, "SHE". 2: Structure of a physically-based, distributed modelling system, J. Hydrol., 87, 61-77.

Abdul, A. S. and R. W. Gillham (1989), Field studies of the effects of the capilary fringe on streamflow generation, J. Hydrol., 112, 1-18.

Abelson, Harold and Gerald J. Sussman (1985), Structure and Interpretation of Computer Programs. MIT Press, Cambridge, Massachusetts.

Anderson M. G. and T. P. Burt (1990a), Process studies in hillslope hydrology: an overview, part of Process Studies in Hillslope Hydrology, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Anderson M. G. and T. P. Burt (1990b), Subsurface runoff, part of Process Studies in Hillslope Hydrology, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Askew, Arthur J. (1970), Variation in lag time for natural catchments, J. Hydraul. Div., Am, Soc. Civ. Engrs., 96 (HY2), 317-330.

Band, L. (1986), Topographic partition of watersheds with digital elevation models, Wat. Resour. Res., 22 (1), 15-24.

Bathurst, James C. (1986), Physically-based distributed modelling of an upland catchment using the Systeme Hydrologique Europeen, J. Hydrol., 87, 79-102.

Bathurst, James C. (1986), Sensitivity analysis of the Systeme Hydrologique Europeen for an upland catchment, J. Hydrol., 87, 103-123.

Becchi, Ignazio, Enrica Caporali and Elena Palmisano (1992) Hydrological response to radar rainfall maps through a distributed model, unpublished manuscript.

Beven, Keith (1979), On the generalized kinematic routing method, Wat. Resour. Res., 15 (3), 1238-1242.

Beven, Keith (1981), Kinematic subsurface stormflow, Wat. Resour. Res., 17 (5), 1419-1424.

Beven, Keith (1982a), On subsurface stormflow: an analysis of response times, Hydrol. Sci. J., 4 (12), 505-521.

Beven, Keith (1982b), On subsurface stormflow: predictions with simple kinematic theory for saturated and unsaturated flows, Wat. Resour. Res., 18 (6), 1627-1633.

Beven, Keith (1984), Infiltration into a class of vertically non-uniform soils, Hydrol. Sci. J., 29 (4), 425-434.

Beven, K. (1985), Distributed models, part of Hydrological Forecasting, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Beven, Keith (1989), Changing ideas in hydrology - The case of physically-based models, J. Hydrol., 105, 157-172.

Beven, Keith, Kevin Gilman and Malcolm Newson (1979), Flow and flow routing in upland channel networks, Hydrol. Sci. Bull., 24 (3), 303-325.

Beven, K. J. and M. J. Kirkby (1979), A physically-based, variable contributing area model of basin hydrology, Hydrol. Sci. Bull., 24 (1), 43-69.

Beven, Keith and Eric F. Wood (1983), Catchment geomorphology and the dynamics of runoff contributing areas, J. Hydrol., 65, 139-158.

Binley, A. M. and K. J. Beven (1991), Physically-based modelling of catchment hydrology: a likelihood approach to reducing predictive uncertainty, part of Computer Modelling in the Environmental Sciences (Proceedings of the Keyworth conference), D. G. Farmer and M. J. Rycroft, editors . Clarendon Press, Oxford.

Binley, Andrew, John Elgy and Keith Beven (1989a), A physically based model of heterogeneous hillslopes. 1. Runoff prediction, Wat. Resour. Res., 25 (6), 1219-1226.

Binley, Andrew, Keith Beven and John Elgy (1989b), A physically based model of heterogeneous hillslopes. 2. Effective hydraulic conductivities, Wat. Resour. Res., 25 (6), 1227-1233.

Blain, C. A. and P. C. D. Milly (1990), Development and application of a hillslope hydrologic model, Adv. Water Resources, 14 (4), 168-174.

Booch, Grady (1991), Object Oriented Design with Applications. Benjamin Cummings Publishing Company, Inc.

Brooks R. H. and A. T. Corey (1964), Hydraulic properties of porous media, Hydrology Paper 3, Colorado State University, Fort Collins, Colorado.

Boyd, Michael J., David H. Pilgrim and Ian Cordery (1979), A storage routing model based on catchment geomorphology, J. Hydrol., 42, 209-230.

Brown, David C. and B. Chandrasekaran (1989), Design Problem Solving. Knowledge Structures and Control Strategies, Research Notes in Artificial Intelligence, Morgan Kauffmann Publishers, Inc.

Cabral, Mariza C., Rafael L. Bras, David Tarboton and Dara Entekhabi (1990), A distributed, physically-based, rainfall-runoff model incorporating topography for real-time flood forecasting, Ralph M. Parsons Lab. Report N. 332. M. I. T. Dep. of Civil Engineering.

Cabral, M. C., L. Garrote, R. L. Bras and D. Entekabi (1992), Kinematic infiltration in vertically heterogeneous, anisotropic and sloped soils, Accepted for publication in Advances in Water Resources.

Campbell, Steven D. and Stephen H. Olson (1986), WX1, an expert system for weather radar interpretation, part of Coupling Symbolic and Numerical Computing in Expert Systems, J. S. Kowalik (ed.), North-Holland, Elsevier Science Publishers.

Charbeneau, Randall J. (1984), Kinematic models for soil moisture and solute transport, Wat. Resour. Res., 20 (6), 699-706.

Childs, E. C. and M. Bybordi (1969), The vertical movement of water in stratified porous material. 1. Infiltration, Wat. Resour. Res., 5 (2), 446-459.

Church, Michael and Ming-Ko Woo (1990), Geography of surface runoff: some lessons for research, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Collinge, V. K. and C. Kirkby (eds.) (1987), Weather radar and flood forecasting, John Wiley & Sons.

Cooper, S. P. and W. A. Norton (1991), Towards an improved person-machine interface in atmospheric modelling, part of Computer Modelling in the Environmental Sciences (Proceedings of the Keyworth conference), D. G. Farmer and M. J. Rycroft, editors . Clarendon Press, Oxford.

Cox, Brad J. (1986), Object-Oriented Programming: An Evolutionary Approach, Addison Wesley.

Dunne, T. and Black, R. D. (1970a), An experimental investigation of runoff production in permeable soils, Wat. Resour. Res., 6 (2), 478-490.

Dunne, T. and Black, R. D. (1970b), Partial area contributions to storm runoff in a small New England watershed, Wat. Resour. Res., 6 (5), 1296-1311.

Dunne, T., T. R. Moore and C. H. Taylor (1975), Recognition and prediction of runoff producing zones in humid regions, Hydrol. Sci. Bull., 20, 305-327.

Fedra, Kurt and Daniel P. Loucks (1985), Interactive computer technology for planning and policy modeling, Wat. Resour. Res., 21 (2), 114-122.

Freeze, R. Allan (1971), Three-dimensional, transient, saturated-unsaturated flow in a groundwater basin, Wat. Resour. Res., 7(2), 347-366.

Freeze, R. Allan (1972), Role of subsurface flow in generating surface runoff. 2. Upstream source areas, Wat. Resour. Res., 8 (5), 1272-1283.

Freeze, R. Allan (1980), A stochastic-conceptual analysis of rainfall-runoff processes on a hillslope, Wat. Resour. Res., 17 (2), 431-432.

Gan, Thian Y. and Stephen J. Burgess (1990a), An assessment of a conceptual rainfall-runoff model's ability to represent the dynamics of small hypothetical catchments. 1. Models, model properties and experimental design, Wat. Resour. Res., 26 (7), 1595-1604.

Gan, Thian Y. and Stephen J. Burgess (1990b), An assessment of a conceptual rainfall-runoff model's ability to represent the dynamics of small hypothetical catchments. 2. hydrologic responses for normal and extreme rainfall, Wat. Resour. Res., 26 (7), 1605-1619.

Gbureck, William J. (1990), Initial contributing area of a small watershed, J. Hydrol., 118, 387-403.

Germann, Peter F. (1990), Macropores and hydrologic hillslope processes, part of Process Studies in Hillslope Hydrology, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Ghezzi, Carlo, Mehdi Jazayeri and Dino Mandrioli (1991), Fundamentals of Software Engineering, Prenticel Hall, Englewood Cliffs, New Jersey.

Goodrich, David C. and David A. Woolhiser (1991), Catchment hydrology, Reviews of Geophisics, Supplement April 1991 (U.S. National Report to International Union of Geodesy and Geophysics 1987-1990), 202-209.

Gorlen, Keith E., Sanford M. Orlow and Perry S. Plexico (1990), Data Abstracion and Object-Oriented Programming in C++, John Wiley and Sons.

Hebbert R. H. B. and R. E. Smith (1990), Hillslope parameter estimation using the inverse procedure, J. Hydrol., 119, 307-334.

Hino, Mikio, Yoshio Odaka, Kazuo Nadakoa and Akito Sato (1988), Effect of initial soil moisture content on the vertical infiltration process - A guide to the problem of runoff-ratio and loss, J. Hydrol., 102, 267-284.

Hornberger, G. M., K. J. Beven, B. J. Cosby and D. E. Sappington (1985), Shenandoah Watershed study: Calibration of a topography-based, variable contributing area hydrological model to a small watershed, Wat. Resour. Res., 21 (12), 1841-1850.

Hornberger, George M., Peter F. Germann and Keith Beven (1991), Throughflow and solute transport in an isolated sloping soil block in a forested catchment, J. Hydrol., 124, 81-99.

James, Wesley P. and Keu Whan Kim (1990), A distributed dynamic watershed model, Water Resources Bull., 26 (4), 587-596.

Johnson, Lynn E. (1986), Men, models, methods, and machines in hydrologic analysis, Journal of Water Resources Planning and Management, Am, Soc. Civ. Engrs., 112 (3), 308-325.

Johnson, Lynn E. (1989), MAPHYD - A digital map-based hydrologic modeling system, Photogrametric Engineering and Remote Sensing, 55 (6), 911-917.

Jones, Oliver (1989), Introduction to the X Window System, Prentice-Hall, Englewood Cliffs, New Jersey.

Kellerhalls, Rolf (1970), Runoff routing thorugh steep natural channels, J. Hydraul. Div., Am, Soc. Civ. Engrs., 96 (HY11), 2201-2217.

Kernighan, Brian W. and Dennis M. Ritchie (1988), The C Programming Language, Prenticel Hall, Englewood Cliffs, New Jersey.

Kim, Won (1990), Introduction to Object-Oriented Databases. MIT Press, Cambridge Massachusetts.

Kirkby, Mike (1986), A runoff simulation model based on hillslope topography, part of Scale Problems in Hydrology, V. K. Gupta, I. Rodrîguez-Iturbe and E. F. Wood, eds., Reidel Publishing Company.

Leopold, L. B. and T. Maddock (1953), The hydraulic geometry of stream channels and some physiographic implications, U.S. Geol. Surv. Prof. Paper No. 252.

Loague, Keith M. (1988), Impact of rainfall and soil hydraulic property information on runoff predictions at the runoff scale, Wat. Resour. Res., 24 (9), 1501-1510.

Loague, Keith and Gene A. Gander (1990), R-5 revisited. 1. Spatial variability of infiltration on a small rangeland catchment, Wat. Resour. Res., 26 (5), 957-971.

Loucks, Daniel P., Janusz Kindler and Kurt Fedra (1985), Interactive water resources modeling and model use: An overview, Wat. Resour. Res., 21 (2), 95-102.

Mancini, M., C. Paniconi and E. F. Wood (1992), Comparison of detailed and conceptual models for the simulation of hillslope flow processes, Eos Trans. AGU, 73 (14), 138.

McCord, James T. and Daniel B. Stephens (1987), Lateral moisture flow beneath a sandy hillslope without an apparent impending layer, Hydrological Processes, 1.

McDonnell J. J., M. Bonell, M. K. Stewart and A. J. Pearce, Deuterium variations in storm rainfall: Implication for stream hydrograph separation, Wat. Resour. Res, 26 (3), 455-458

Mein, Russell G., Eric M. Laurenson and Thomas A. McMahon (1974), Simple nonlinear model for flood estimation, J. Hydraul. Div., Am, Soc. Civ. Engrs., 100 (HY11), 1057-1518.

Minshall, N. E. (1960), Predicting storm runoff on small experimental watersheds, J. Hydraul. Div., Am, Soc. Civ. Engrs., 86 (HY8), 17-38.

Moore, I. D., R. B. Grayson and A. R. Larson (1991), Digital terrain modelling: a review of hydrological, geomorphological and biological applications, Hydrological Processes, 5, 3-30.

Moore, I. D., R. B. Grayson and J. P. Wilson (1990), Runoff modelling in complex three-dimensional terrain, part of Hydrology in Mountainous Regions. 1. Hydrological Management of the Water Cycle (Proceedings of two Lausanne Symposia, August 1990). IAHS Publ. no. 193.

Morris, D. G. and Heerdegen, R. G. (1988), Automatically derived catchment boundary and channel networks and their hydrological applications, Geomorphology, 1, 131-141.

Nii, H. Penny (1986), Blackboard systems. Part I: The blackboard model of problem solving and the evolution of blackboard architectures, AI Magazine, 7 (2), 38-53.

Nolan, K. M. and B. R. Hill (1990), Storm-runoff generation in the permanent creek drainage, West Central California - an example of floos-wave effects on runoff composition, J. Hydrol., 113, 343-367.

O'Loughlin E. M. (1981), Saturation regions in catchments and their relations to soil and topographic properties, J. Hydrol., 53, 229-246..

O'Loughlin, E. M. (1990), Perspectives on hillslope research, part of Process Studies in Hillslope Hydrology, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Ott, M., Z. Su, A. H. Schumann and G. A. Schultz (1991), Development of a distributed hydrological model for flood forecasting and impact assessment of land-use change in the international Mosel river basin, part of Hydrology for the Water Management of Large River Basins (Proceedings of the Vienna Symposium, August 1991). IAHS Publ. no. 201.

Philip, J. R. (1969), Theory of infiltration, Adv. Hydrosci., 5, 215-296.

Philip, J.R. (1991a), Hillslope infiltration: planar slopes, Wat. Resour. Res., 27 (1), 109-117.

Philip, J.R. (1991b), Hillslope infiltration: divergent and convergent slopes, Wat. Resour. Res., 27 (6), 1035-1040.

Pilgrim, David H. (1977), Isochrones of travel time and distribution of flood storage from a tracer study on a small watershed, Wat. Resour. Res., 13 (3), 587-595.

Protopapas, Angelos L. and Rafael L. Bras (1991a), The one-dimensional approximation for infiltration in heterogeneous soils, Wat. Resour. Res., 27 (6), 1019-1027.

Protopapas, Angelos L. and Rafael L. Bras (1991b), Analytical solutions for unsteady multidimensional infiltration in heterogeneous soils, Wat. Resour. Res., 27 (6), 1029-1034.

Quinn, P., K. Beven, P. Chevalier and Q. Planchon (1991), The prediction of hillslope flow paths for distributed hydrological modeling using digital terrain models, Hydrological Processes, 5, 59-79.

Roddis, W. M. K. and J. Connor (1988), Qualitative/quantitative reasoning for fatigue and fracture in bridges, part of Coupling Symbolic and Numerical Computing in Expert Systems, II, J. S. Kowalik and C. T. Kitzmiller (eds.), 249-257. North-Holland, Elsevier.

Rogers, C. C. M., K. J. Beven, E. M. Morris and M. G. Anderson (1985), Sensitivity analysis, calibration and predictive uncertainty of the institute of hydrology distributed model, J. Hydrol., 81, 179-191.

Rumbaugh, J., M. Blaha, W. Premerliani, F. Eddy and W. Lorensen (1990), Objetc-Oriented Modeling and Design, Prentice Hall, New Jersey.

Sasowsky, Kathryn C. and Thomas W. Gardner (1991), Watershed configuration and geographic information system parameterization for spur model hydrologic simulations, Water Resources Bull., 27 (1), 7-18.

Scheifer, Robert W. and James Gettys (1991), The X Window system, part of Research Directions in Computer Science, A. R. Meyer, J. V. Guttag, R. L. Rivest and P. Szolovits, eds. MIT Press, Cambridge, Massachusetts.

Seo, D.-J., and J. A. Smith (1992), Radar-based short-term rainfall prediction, J. Hydrol., 131, 341-367.

Singh, Vijay P. (1989), Hydrologic Systems. Volume II, Watershed Modeling. Prentice Hall, Englewood Cliffs, New Jersey.

Skaags, R. W. and R. Khaleel (1982), Infiltration, part of Hydrologic Modeling of Small Watersheds, C. T. Haan, H. P. Johnson and D. L. Brakenskiek (eds.), American Society of Agricultural Engineers.

Smettem K. R. J., D. J. Chittleborough, B. G. Richards and F. W. Leaney (1991), The influence of macropores on runoff generation from a hillslope soil with a contrasting textual class, J. Hydrol., 122, 235-252.

Snyder, Willard M., and John B. Stall (1965), Men, models, methods, and machines in hydrologic analysis, J. Hydraul. Div., Am, Soc. Civ. Engrs., 91 (HY2), 85-99.

Stefik, Mark and Daniel C. Bobrow (1986), Object-oriented programming: Themes and variations, AI Magazine, 6 (4), 40-62.

Stephenson, G. R. and R. A. Freeze (1974), Mathematical simulation of subsurface flow contributions from snowmelt runoff, Reynolds Creek Watershed, Idaho. Wat. Resour. Res., 10(2), 284-298.

Tanaka, T., M. Yasuhara, H. Sakai, and A. Marui (1988), The Hachioji experimental basin study - Storm runoff processes and the mechanism of its generation, J. Hydrol., 102, 139-164.

Tarboton, David (1989), The analysis of river basins and channel networks using digital terrain data, Ph. D. Thesis, M. I. T. Dep. of Civil Engineering.

Tarboton, D., R. L. Bras and I. Rodríguez-Iturbe (1989), Scaling and elevation in river networks, Wat. Resour. Res., 25 (9), 2037-2051.

Tarboton, D., R. L. Bras and I. Rodríguez-Iturbe (1991), On the extraction of channel networks from digital elevation data, Hydrologic Processes, 5 (1), 81-100.

Troendle, C. A. (1985), Variable source area models, part of Hydrological Forecasting, M. G. Anderson and T. P. Burt, eds., John Wiley & Sons.

Widman, Lawrence E., Kenneth A. Loparo and Norman R. Nielsen (eds.) (1989), Artificial intelligence, simulation and modeling, John Wiley and Sons, New York.

Wilhelmij, G. Paul (1991), Symbolic Software for Interactive Descriptions of Dynamic Systems, Research Notes in Artificial Intelligence, Morgan Kauffmann Publishers, Inc.

Wood, Eric F., M. Sivapalan, Keith Beven and Larry Band (1988), Effects of spatial variability and scale with implications to hydrologic modeling, J. Hydrol., 102, 29-47.

Wood, W. L. and A. Calver (1992), Initial conditions for hillslope hydrology modelling, J. Hydrol., 130, 379-397.

Woolhiser, D. A., R. E. Smith and D. C. Goddrich (1990), KINEROS, a kinematic runoff and erosion model: Documentation and user manual. U. S. Dep. of Agriculture.

Young, D. (1989), X Window Systems Programming and Applications with Motif/Xt, Addison Wesley.

Zaslavsky, Dan and A. S. Rogowski (1969), Hydrologic and morphologic implications of anisotropy and infiltration in soil profile development, Soil Sci. Soc. Am. Proc., 33, 594-599.

Zavlasky, Dan and Gideon Sinai (1981a), Surface hydrology: III - Causes of lateral flow, J. Hydraul. Div., Am, Soc. Civ. Engrs., 107 (HY1), 37-52.

Zavlasky, Dan and Gideon Sinai (1981b), Surface hydrology: IV - Flow in sloping, layered soil, J. Hydraul. Div., Am, Soc. Civ. Engrs., 107 (HY1), 53-64.

Zavlasky, Dan and Gideon Sinai (1981c), Surface hydrology: V - In-surface transient flow, J. Hydraul. Div., Am, Soc. Civ. Engrs., 107 (HY1), 65-93.

APPENDIX 1

A Kinematic Model of Infiltration in Vertically Heterogeneous, Anisotropic and Sloped Soils

by

Mariza C. Cabral, Luis Garrote, Rafael L. Bras and Dara Entekhabi*

A1 Introduction

It has long been recognized that infiltrating rainfall on a hillslope results in subsurface flow lines that often are not vertical, but have a downslope component. This paper is motivated by the runoff-generation implications of subsurface flow geometry, with flood forecasting as the ultimate goal. Lateral subsurface flux originating during a rainfall event may generate storm runoff by two different mechanisms. First, lateral flux may emerge at the surface under certain conditions, becoming overland flow. This is subsurface storm runoff and it may represent a significant portion of the storm hydrograph. Second, lateral subsurface flux leads to moisture concentration at the bottom of the hillslopes, especially if the hillslopes are concave and convergent. Commonly the bottom of these hillslopes are also areas of shallow water table, and therefore subsurface inputs to these areas, combined with direct rainfall,

* This appendix is a reprint of a paper describing the local infiltration model used in the Distributed Basin Simulator. The paper is accepted for publication in *Advances in Water Resources.*

cause rapid water table rise and the development and growth of saturated areas that are effectively impermeable to rainfall. From this perspective, topography plays an important role in storm runoff generation. The availability of detailed topographical information in the form of digital elevation models (DEM) facilitates the use of topography in a distributed flood simulation model. However, in order to obtain a distributed model that can operate at the fine resolution given by DEM's it is necessary to develop a computationally efficient model of infiltration that can represent the most relevant features of the subsurface flow from the standpoint of storm runoff generation. The infiltration model should nevertheless include the role of sloped terrains, anisotropic soil layering and other factors distributing moisture in the vertical soil column.

Capillary forces, soil layering, anisotropy and slope are the main mechanisms through which flow deviations from the vertical have been explained in the literature. *Zaslavsky and Rogowski* [1969] introduce some qualitative and quantitative considerations in modeling the influence of anisotropy and terrain slope on the magnitude and direction of unsaturated downhill flow. *Zaslavsky and Sinai* [1981] study the effects of soil heterogeneity, specifically soils consisting of distinct and plane-parallel layers. Successive layers of homogeneous and isotropic soil with different hydraulic properties behave as a nonisotropic soil matrix with a resultant downstream flow component parallel to the soil surface. *McCord and Stephens* [1987] analyze soil moisture data from an experimental basin in order to characterize the influence of topography on subsurface lateral flow paths in the unsaturated zone.

An infinite-series solution to the full equation of flow in a homogeneous anisotropic sloped soil is presented by *Philip* [1991]. He

finds that total horizontal unsaturated flow is generally directed into the hillslope for isotropic soils. The integrated resultant flow, nevertheless, has a constant downslope component which is proportional to the square root of time for small times. He also considers two types of anisotropy. In the first type the principal directions of anisotropy are parallel and normal to the soil surface. In the second type, principal directions of anisotropy are horizontal and vertical. *Philip* [1991] finds markedly different behavior in both cases, depending on which component of the flow obtained for the isotropic case, the parallel or the horizontal, is magnified by the anisotropy ratio.

Series and numerical solutions to the flow equation provide a good insight to the main features of the physical description of the phenomenon, but they are computationally too complex to be included in distributed basin-scale models. In this paper, several simplifying assumptions are made in order to obtain a simple analytical solution feasible for application at the basin scale. Runoff generation during floods is of concern, and therefore an adequate mapping of the saturated areas is the main objective of the model. Rainfall over these saturated subregions is responsible for a large fraction of the peak in the storm hydrograph.

The case analyzed here is similar to that in *Philip* [1991], considering anisotropy parallel to the hillslope for a particular case of soil heterogeneity. The simplification introduced consists of neglecting the effect of capillary forces in unsaturated infiltration in what is usually referred to as the kinematic approximation. Kinematic infiltration has been studied in the context of gravity-dominated flow in homogeneous and isotropic soils [*Beven et al.*, 1981; *Smith and Herbert*, 1983; *Charbeneau*, 1984]. It has been found to be an adequate approximation to obtain

subsurface stormflow by *Beven* [1981] for a wide range of values of hydraulic conductivity and terrain slope. Here we consider kinematic infiltration in anisotropic soils within sloping surfaces where the saturated hydraulic conductivity decreases exponentially with depth. For the sake of mathematical tractability, it is assumed that the soil column has a large length scale in the direction of its surface slope so that boundary conditions do not have an effect on the local infiltration. *Philip* [1991] finds this assumption not too restrictive for most practical cases. However, this assumption should be revised for application in a distributed model, since the spatial variability of hillslope geometry and soil properties introduces lateral flow gradients . Here we are concerned with one dimensional infiltration only; therefore variabilities in the direction parallel to the slope will not be considered.

A1.2 Soil Model

Flow equations are derived in the reference system defined by the directions normal (n) and parallel (p) to the slope; $\mathbf{i_n}$ and $\mathbf{i_p}$ are unit vectors in these directions. The distance n is taken positive in the downward direction and p is taken positive in the downslope direction. Flow is also described in the alternative reference system defined by the vertical (z) and horizontal (x) directions. $\mathbf{i_z}$ and $\mathbf{i_x}$ are unit vectors in these directions. The two reference systems form an angle α, the angle of terrain slope (see Figure 1).

The analytical solutions to the kinematic infiltration are developed for the following model of the soil column.

1) The soil is sloped at an angle α with respect to the horizontal direction. Soil characteristics may vary only in the direction normal to the surface. The terrain model is therefore layered parallel to the surface.

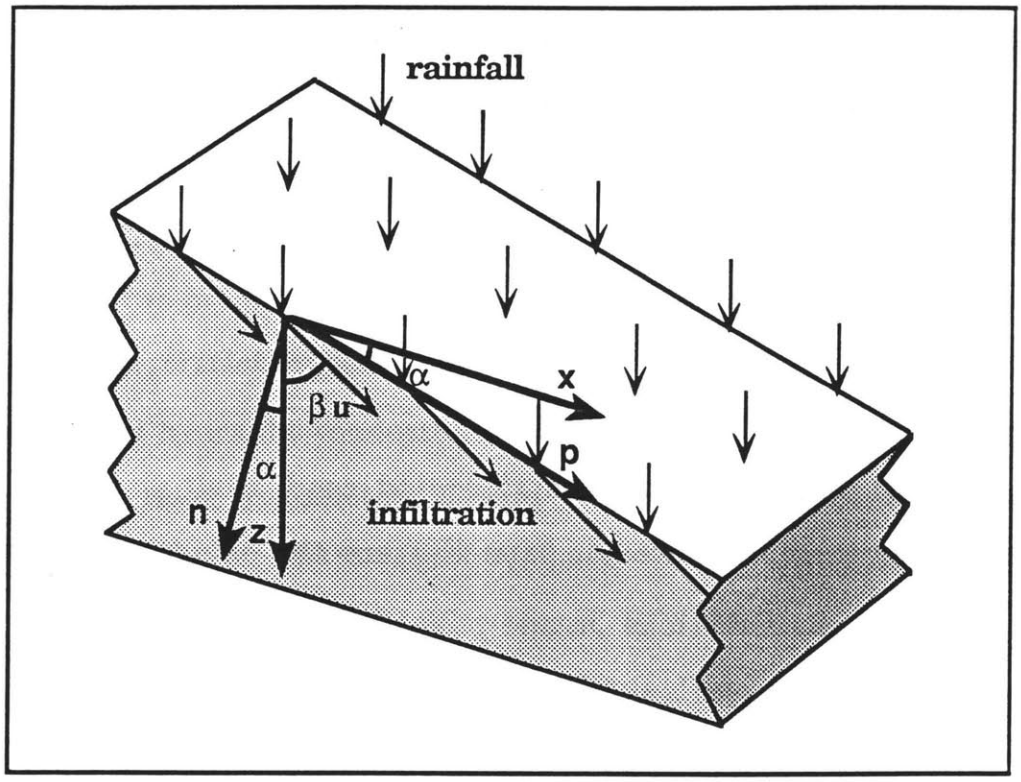


Figure 1: *Representation of the coordinate directions on a hillslope of constant slope, tan(α).*

2) Saturated hydraulic conductivity decreases with depth normal to the sloped surface. The decrease of hydraulic conductivity with depth is a key assumption in the model, since it is the mechanism that leads to the formation of a perched saturated zone. Different functional forms could be adopted to parameterize the decrease of hydraulic conductivity with depth. Here we adopt an exponential decrease with normal depth. *Beven* [1982] finds that a number of soil data sets from a variety of basins were

well represented by such a model of hydraulic conductivity. It is important to note that results can be obtained for other parameterizations for the decrease of saturated conductivity with depth.

The directions parallel and normal to the soil surface are the main directions of the hydraulic conductivity tensor, and for them, the saturated hydraulic conductivities are given by

$$K_{S_n}(n) = K_{0_n} e^{-fn} \tag{1a}$$

$$K_{S_p}(n) = K_{0_p} e^{-fn} \tag{1b}$$

where $K_{sn}(n)$ and $K_{sp}(n)$ are the saturated conductivities at depth n perpendicular to the surface; K_{0n} and K_{0p} are the saturated hydraulic conductivities in directions *n* and *p* at the soil surface; and f is a parameter of dimension [L^{-1}]. The parameter f controls the decay of the saturated hydraulic conductivity with depth.

The saturated hydraulic conductivities in directions *n* and *p* are related through the dimensionless anisotropy ratio a_r, defined as

$$a_r = \frac{K_{0_p}}{K_{0_n}} > 1 \tag{2}$$

This relationship is assumed to be valid for all depths.

3) The Brooks-Corey [*Brooks and Corey*, 1964] parameterization of unsaturated hydraulic conductivity is used. Upon substitution of Equations (1a) and (1b) for the saturated conductivities in directions *n* and *p*, the Brooks-Corey parameterization gives,

$$K_n(\theta,n) = K_{0_n} e^{-fn} \left(\frac{\theta - \theta_r}{\theta_s - \theta_r} \right)^{\varepsilon} \tag{3a}$$

$$K_p(\theta,n) = K_{0_p} e^{-fn} \left(\frac{\theta - \theta_r}{\theta_s - \theta_r} \right)^{\varepsilon} \tag{3b}$$

where $K_n(\theta,n)$ and $K_p(\theta,n)$ are the hydraulic conductivities in directions *n* and *p* at moisture content θ and at depth n; θ_S is the saturated moisture content; θ_r is the residual moisture content, defined as the value below which moisture is immobile; and ε is a pore size distribution index.

There is correlation between each of the parameters θ_r, θ_s, and ε and the saturated hydraulic conductivity. For saturated conductivity varying with depth, these parameters should correspondingly be functions of depth. Nevertheless, parameters θ_r, θ_s, and ε have less variability for different soil types [*Clapp and Hornberger*, 1978; *Mualem*, 1978], and for simplicity we consider them to be constant with depth. The same approach has been adopted by other authors [*Dagan and Bressler*, 1983; *Yeh et al*, 1985; etc.]. While saturated hydraulic conductivity varies over many orders of magnitude in heterogeneous soils, the pore size distribution index for the same soils varies well below even one order of magnitude. Given these considerations, it is evident that anisotropy of saturated conductivities is duplicated for unsaturated flow as well.

A1.3 Analysis of unsaturated flow

When the rainfall rate is lower than the initial infiltration capacity, the movement of water in the soil column occurs under unsaturated conditions. In a soil where conductivity decreases with depth, perched saturation may develop under prolonged infiltration. All equations presented in this Section refer to flow in the unsaturated region from the surface to the wetting front in the early stages of infiltration. Flow under saturated conditions is the subject of Section 4.

A1.3.1 Flow geometry

The flow vector **q** may be expressed in term of its components in the main directions of anisotropy ,

$$\mathbf{q} = q_n \mathbf{i}_n + q_p \mathbf{i}_p = - K_n J_n \mathbf{i}_n - K_p J_p \mathbf{i}_p \qquad (4)$$

where q_n and q_p are the components of discharge per unit area in directions *n* and *p* and J_n and J_p are the components of the hydraulic gradient vector **J** in those directions. If the soil is unsaturated, water pressure is less than atmospheric, because it is affected by surface tension. However, under the kinematic assumption the effect of negative water pressure due to capillarity is neglected and the hydraulic gradient is only gravitational,

$$\mathbf{J} = J_n \mathbf{i}_n + J_p \mathbf{i}_p = - \cos(\alpha) \mathbf{i}_n - \sin(\alpha) \mathbf{i}_p \qquad (5)$$

Substituting J_n and J_p given by Equation (5) into Equation (4), and using Equation (2) we write

$$\mathbf{q} = K_n \cos(\alpha)\, \mathbf{i_n} + K_p \sin(\alpha)\, \mathbf{i_p} = K_n \cos(\alpha)\, \mathbf{i_n} + a_r K_n \sin(\alpha)\, \mathbf{i_p} \quad (6)$$

Soil anisotropy deflects the unsaturated flow at an angle, designated β_u, with respect to the direction of the hydraulic gradient (the subscript in β_u is used to indicate unsaturated infiltration). From Figure 2 and Equations (2), (4) and (6),

$$\tan(\alpha + \beta_u) = \frac{q_p}{q_n} = a_r \tan(\alpha) \quad (7)$$

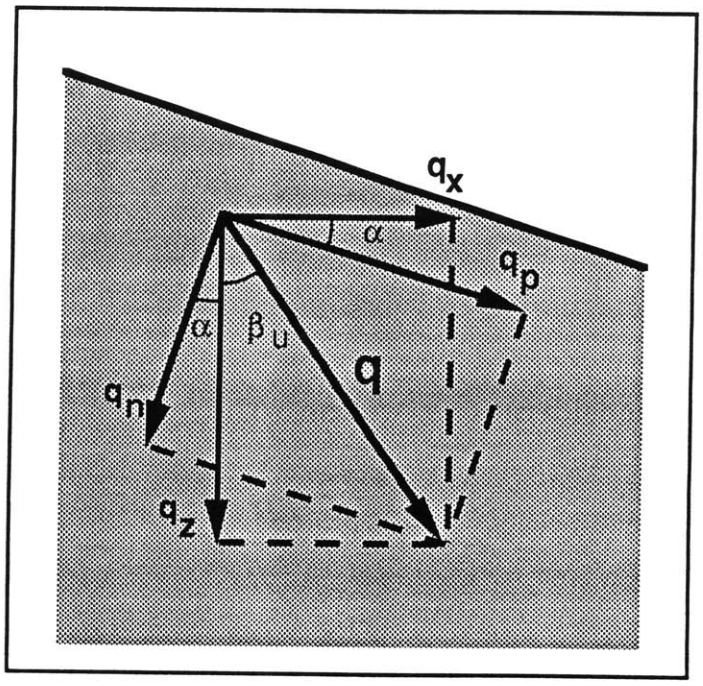


Figure 2: *Components of flow in the z and x, and p and n directions. Flow is in the direction indicated by the vector* $\boldsymbol{q}$.

Solving for β_u,

$$\beta_u = \tan^{-1}(a_r \tan(\alpha)) - \alpha \qquad (8)$$

Angle β_u is constant with depth since the ratio a_r is constant with depth. It is larger for higher anisotropy ratios and steeper terrain slopes. For $a_r = 1$, Equation (8) gives β_u equal to zero. That is, for an isotropic soil the above equations give unsaturated flow strictly in the vertical direction.

Since flow direction is constant in the unsaturated area, we can derive an expression for the steady state flow vector **q** from continuity considerations. Let us consider rainfall at a rate R. In steady state there must be no moisture changes in the soil. From continuity, the flow per unit width perpendicular to segment *a* (which has unit length) must equal the flow per unit width perpendicular to segment *b* in Figure 3.

Designating the direction of the resultant flow vector by *s*,

$$R\, a = q_s\, b \qquad (9)$$

From geometrical considerations, we can write

$$b = \frac{\cos(\alpha + \beta_u)}{\cos(\alpha)} \qquad (10)$$

Substituting (9) into (10) and solving for q_s we obtain

$$q_s = \frac{\cos(\alpha)}{\cos(\alpha + \beta_u)} R \qquad (11)$$

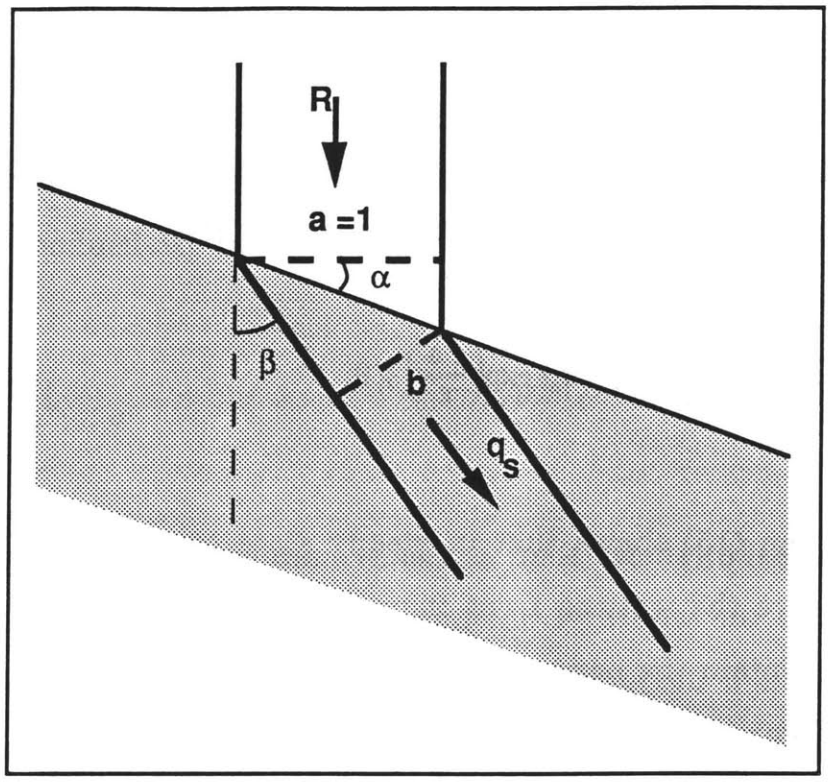


***Figure 3**: Unsaturated infiltration in a laterally anisotropic soil.*

The component of flow in the normal direction q_n, is related geometrically to q_s through

$$q_n = \cos(\alpha + \beta_u)\, q_s \qquad (12)$$

which combines with Equation (11) into

$$q_n = R \cos(\alpha) \qquad (13)$$

From Equations (13) and (6) we obtain

$$K_n(\theta,n) = R \qquad (14)$$

which is evident for kinematic flow in porous media.

Using the above in Equation (3a) we obtain a description of the soil moisture profile under steady infiltration at rate R

$$\theta(R,n) = \left(\frac{R}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r) \exp\left(\frac{f\,n}{\varepsilon}\right) + \theta_r \qquad (15)$$

Equation (15) shows that in steady state conditions, the moisture content above the wetting front increases exponentially with depth, in order to maintain the normal hydraulic conductivity equal to the rainfall rate R. Figure 4 represents moisture profiles corresponding to various steady infiltration rates.

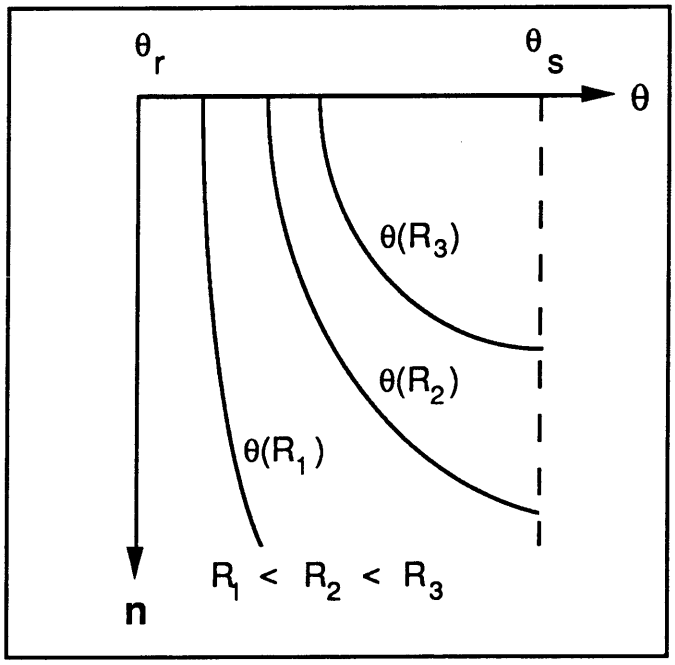


***Figure 4**: Moisture profiles for various rainfall rates lower than the soil surface saturated conductivity.*

Given that saturated soil conductivity decreases with depth, for any rainfall rate R lower than the surface normal hydraulic conductivity ($R < K_{0_n}$), a corresponding depth within the soil profile N*(R) will have the saturated conductivity in the normal direction equal to the rainfall rate, i.e.

$$K_n(\theta_s, N^*) = R \tag{16}$$

Substituting θ_s and N* in the expression for $K_n(N^*)$ (Equation (3a)) and solving for N* in Equation (16) we obtain,

$$N^*(R) = \frac{1}{f} \ln\left(\frac{K_{0_n}}{R}\right) \tag{17}$$

Note that Equation (17) applies only for $R \leq K_{0n}$. For $R > K_{0n}$, the saturated level is at the surface, and there is no unsaturated area above the wetting front.

Letting n = N*(R) in Equation (15), we obtain $q(R,N^*) = q_s$. Therefore, N*(R) represents the depth at which saturation develops under a steady infiltration rate R. For $n > N^*(R)$, we have $K_{s_n}(n) < R$ and the soil can no longer transmit flow at the rate of infiltration to depths beyond N*(R). Water accumulates above that level and perched saturation develops.

A1.3.2 Vertical and horizontal components of flow

Combining Equations (6), (7), (13), and (16) the expression for the flow vector **q** is

$$\mathbf{q} = R \cos(\alpha)\, \mathbf{i}_n + a_r R \sin(\alpha)\, \mathbf{i}_p \qquad (18)$$

Vertical and horizontal components of flow are obtained by projecting the flow vector **q** on the z and x axes:

$$q_z = q_n \cos(\alpha) + q_p \sin(\alpha) = [R \cos(\alpha)] \cos(\alpha) + [a_r R \sin(\alpha)] \sin(\alpha) \qquad (19a)$$

$$q_x = -q_n \sin(\alpha) + q_p \cos(\alpha) = -[R \cos(\alpha)] \sin(\alpha) + [a_r R \sin(\alpha)] \cos(\alpha) \qquad (19b)$$

Upon simplification,

$$q_z = R [\cos^2(\alpha) + a_r . \sin^2(\alpha)] \qquad (20a)$$

$$q_x = R \cos(\alpha) \sin(\alpha) (a_r - 1) \qquad (20b)$$

Both vertical and horizontal components of flow are constant over depth. For anisotropy ratios greater than one, horizontal flow goes in the downhill direction and is maximum for slopes at 45° angle .

A1.3.3 Wetting front advance in the unsaturated zone

Equations relevant to the geometry and magnitude of the unsaturated flow have been derived under the steady state assumption. In this section we present the equation for the time evolution of the moisture front in the soil column. We assume that the initial state of the soil can be described by a moisture distribution given by Equation (15) for a small initial infiltration rate R_i. We also consider a constant rainfall rate R, $R > R_i$,

over a given time interval. Under the kinematic approximation, a sharp discontinuity in moisture content separates the area affected by the propagation of the infiltration wave and the undisturbed area below the front.

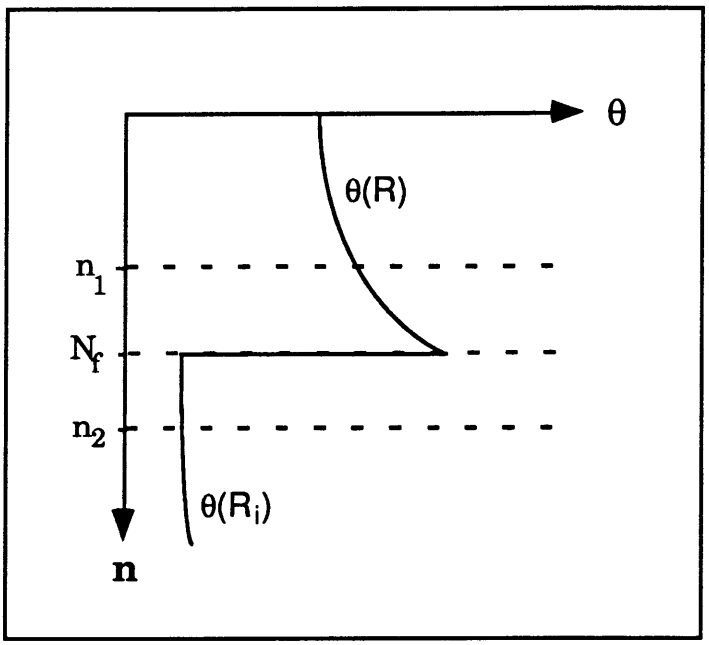


***Figure 5**: Wetting front described by the kinematic model of infiltration.*

We consider that the expression governing the flow of moisture in the unsaturated area in steady state (Equation (18)) is valid for both areas just above and below the wetting front. Furthermore, we assume that the wetting front is parallel to the surface and advances perpendicular to it. Its location below the surface datum is represented by the distance N_f (see Figure 5). In the initial phase of infiltration, i.e. while $N_f < N^*(R)$, the soil between the surface and depth N_f is unsaturated. The evolution of N_f is given by

$$\frac{dN_f}{dt} = \frac{(R - R_i)\cos(\alpha)}{\theta(R\ ,\ N_f) - \theta(R_i,\ N_f)} \qquad (21)$$

which is derived in the Appendix (Section A1.9).

The rate of advance of the wetting front depends on the difference between the rainfall rate and the initial recharge rate ($R - R_i$) and the difference between the moisture distributions corresponding to R and R_i at the depth of the wetting front. Once the wetting front reaches the critical depth $N^*(R)$, soil conductivity will equal the infiltration rate. From that point downwards, the soil can no longer increase its moisture content to keep normal hydraulic conductivity equal to the rainfall rate, and therefore perched saturation develops at and above the wetting front. Flow in this saturated zone is the subject of next Section.

A1.4 Analysis of saturated flow

The soil column above the wetting front is saturated in two cases; 1) when the rainfall rate is higher than the surface saturated conductivity ($R > K_{0n}$), or 2) when the wetting front has penetrated beyond the critical depth ($N_f \geq N^*(R)$).

In the first case, the entire wetted soil (i.e., from the surface to N_f) is saturated. In the second case, as the front reaches $N^*(R)$ infiltration can only be less than recharge from above, and moisture progressively accumulates above the wetting front. A zone of perched saturation develops and grows upward from $N^*(R)$, as well as downward as the front progresses. In order to fully describe the soil moisture profile in this case,

it is necessary to specify not only the depth of the wetting front but also the depth of the top of the zone of perched saturation. Therefore, we introduce a new variable N_t, defined as N_f minus the height of the zone of perched saturation. If there is a zone of perched saturation, this definition corresponds to the normal distance from the top of the zone of saturation to the soil surface. If the whole wetted soil is unsaturated ($N_f < N^*(R)$), then N_t equals N_f. Equations of evolution for N_t as well as for N_f are derived below. Figure 6 represents the progression of the wetting front in time, showing the development of a zone of saturation and the upward evolution of N_t.

A1.4.1 Flow geometry

The first term in the continuity equation (A1) is zero within the zone of saturation since $\theta(t) = \theta_s$ and the saturated flow is laterally non-divergent. Since we have also assumed that all derivatives in the *p* direction are zero, the non-divergence of flow translates into

$$\frac{\partial q_n}{\partial n} = 0 \qquad (22)$$

Equation (22) yields $q_n(n)$ as constant in the *n* direction. Since the elevation gradient is constant and hydraulic conductivity decreases with depth, constant normal flow within the saturated zone implies a positive pressure buildup within it. Pressure gradient compensates for the different hydraulic conductivities of the successive layers of the saturated zone in order to keep normal flow constant. Therefore, the hydraulic

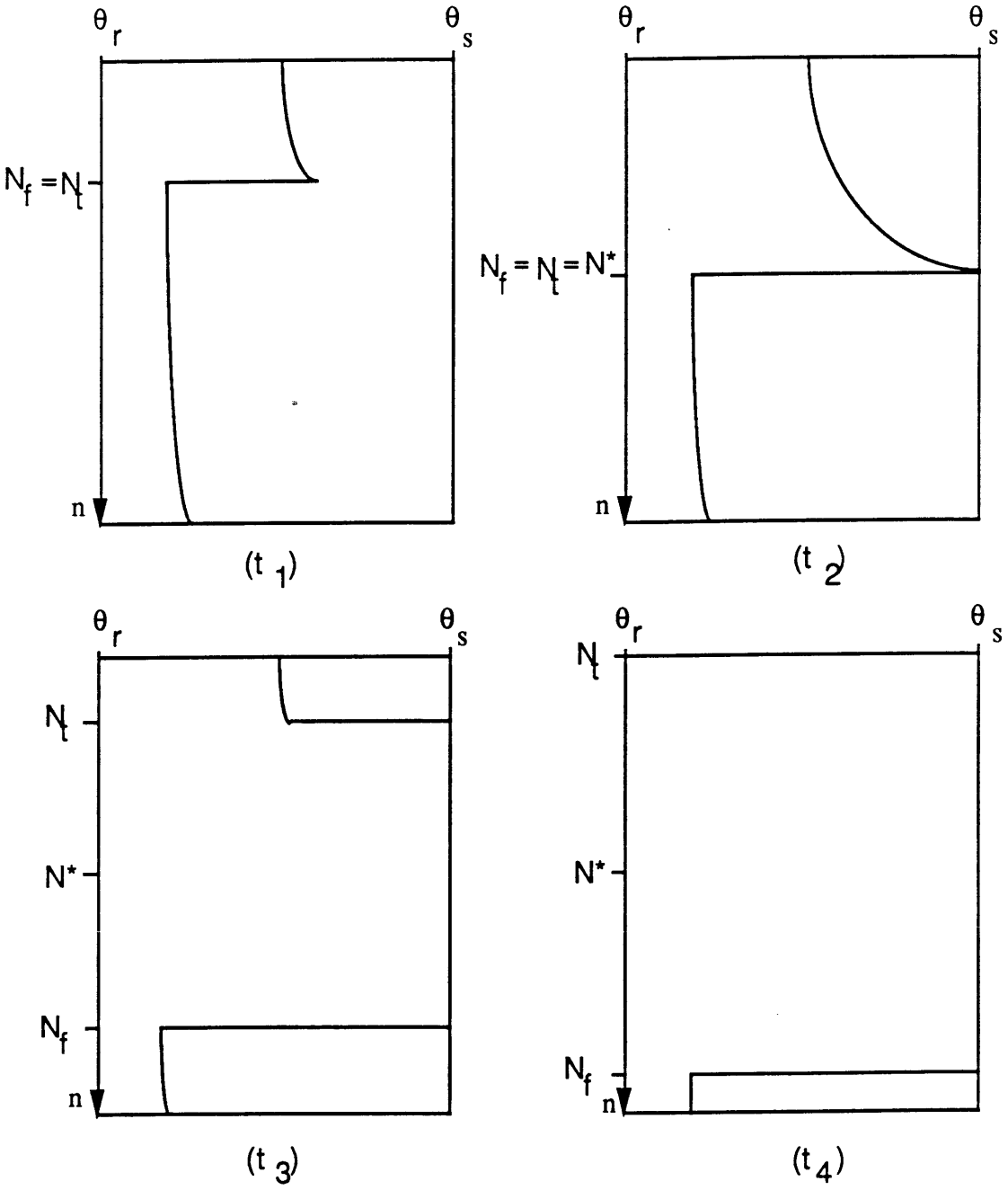


***Figure 6**: Evolution of the moisture profile over four consecutive time steps, t_1 through t_4, under constant rainfall rate R.*

gradient within the saturated zone has to account for the gradient of that positive pressure distribution,

$$\mathbf{J} = J_n\ \mathbf{i_n} + J_p\ \mathbf{i_p} = \left(-\cos(\alpha) + \frac{\partial \Psi}{\partial n}\right)\mathbf{i_n} - \sin(\alpha)\ \mathbf{i_p} \qquad (23)$$

where scalar Ψ is a positive pore pressure, which varies only with normal depth alone.

The flow equation in the saturated zone is,

$$\mathbf{q} = q_n\ \mathbf{i_n} + q_p\ \mathbf{i_p} = -K_{s_n} J_n\ \mathbf{i_n} - K_{s_p} J_p\ \mathbf{i_p} = -K_{0_n} e^{-fn} J_n\ \mathbf{i_n} - K_{0_p} e^{-fn} J_p\ \mathbf{i_p} \qquad (24)$$

Substituting the value of the hydraulic gradient in the *n* direction J_n into the expression of q_n,

$$q_n(n) = \left(\cos(\alpha) - \frac{\partial \Psi}{\partial n}\right) K_{0_n} e^{-fn} \qquad (25)$$

Upon substitution of Equation (25) into Equation (22) we obtain the differential equation that governs pressure distribution within the saturated zone,

$$\frac{\partial}{\partial n}\left[\left(\cos(\alpha) - \frac{\partial \Psi}{\partial n}\right) K_{0_n} e^{-fn}\right] = 0$$

or

$$\frac{\partial^2 \Psi}{\partial n^2} - f\frac{\partial \Psi}{\partial n} + f\cos(\alpha) = 0 \qquad (26)$$

Pressure at both fronts N_t and N_f can be assumed atmospheric given that they are in contact with the unsaturated zones, which have zero pressure according to the kinematic model of infiltration. The area above the top front is directly in contact with the atmosphere. The area below the wetting front can be assumed to be open to atmospheric pressure given the spatial variability in the real moisture distribution at the basin scale. Integration of Equation (26) with the boundary conditions $\Psi(N_f) = \Psi(N_t) = 0$ leads to

$$\Psi(n) = \cos(\alpha)\left[\left(n + \frac{1}{f}\right) - \frac{e^{fN_f} - e^{fn}}{e^{fN_f} - e^{fN_t}}\left(N_t + \frac{1}{f}\right) - \frac{e^{fn} - e^{fN_t}}{e^{fN_f} - e^{fN_t}}\left(N_f + \frac{1}{f}\right)\right] \qquad (27)$$

The hydraulic potential can be obtained upon differentiation of the pressure distribution,

$$\mathbf{J} = -\left(\frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}}\, e^{fn}\right)\cos(\alpha)\,\mathbf{i_n} - \sin(\alpha)\,\mathbf{i_p} \qquad (28)$$

Substitution of J_n in the flow Equation (24) gives the expression for the normal flow in the saturated area,

$$q_n = -K_{0_n}\, e^{-fn}\, J_n = K_{0_n}\,\frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}}\,\cos(\alpha) \qquad (29)$$

Normal flow is constant in the saturated area. We can define an "equivalent hydraulic conductivity" for the saturated area, K_{eq}, as the normal hydraulic conductivity of a homogeneous soil with the same

normal flow q_n given by Equation (29). The equivalent hydraulic conductivity is given by

$$K_{eq}(N_f,N_t) = K_{0_n} \frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \tag{30}$$

This equivalent conductivity also corresponds to the harmonic mean of the conductivities over the saturated depth,

$$K_{eq}\,(N_f,N_t) = \frac{\int_{N_t}^{N_f} dn}{\int_{N_t}^{N_f} \frac{dn}{K_{s_n}(n)}} \tag{31}$$

We may also designate by "equivalent depth", N_{eq}, the normal depth which has conductivity equal to $K_{eq}(N_f,N_t)$. From equations (1a) and (30),

$$N_{eq}(N_f,N_t) = -\frac{1}{f} \ln \left[\frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \right] \tag{32}$$

N_{eq} is also the depth at which the pressure distribution (Equation (27)) is maximum, because for that point the pressure gradient is zero and flow is controlled only by the unit gravitational gradient. For depths smaller than N_{eq}, that is $N_t < n < N_{eq}$, the saturated hydraulic conductivity is greater than K_{eq}, and the pressure gradient is positive (increasing pressure with depth) to compensate for the excess in hydraulic conductivity and to maintain constant normal flow. For depths greater than N_{eq}, that is $N_{eq} < n < N_f$, the saturated hydraulic

conductivity is smaller than K_{eq}, and constant normal flow implies a negative pressure gradient in that area, up to the wetting front, where pressure is again atmospheric.

The parallel component of **J** is entirely gravitational, and the flow in this direction is given by

$$q_p = - K_{0_p} e^{-fn} J_p = K_{0_p} e^{-fn} \sin(\alpha) \tag{33}$$

Within the saturated zone above the wetting front normal flow is constrained by continuity, but parallel flow is not affected by the pressure gradient in the normal direction. As a consequence, flow above the wetting front is deflected laterally. Because conductivity is a function of normal depth n, the angle of flow deflection and the components of flow in directions *p* and *n* are also functions of n. The angle of flow with the vertical direction within the zone of saturation is designated $\beta_s(n)$, and is obtained from (29) and (33),

$$\frac{q_p}{q_n} = \tan(\alpha + \beta_s(n)) = a_r \tan(\alpha) \frac{e^{fN_f} - e^{fN_t}}{f(N_f - N_t)} e^{-fn} \tag{34}$$

and solving for $\beta_s(n)$,

$$\beta_s(n) = \tan^{-1}\left(a_r \tan(\alpha) \frac{e^{fN_f} - e^{fN_t}}{f(N_f - N_t)} e^{-fn} \right) - \alpha \tag{35}$$

Flow deflection with respect to the vertical direction in the saturated area is due to two superimposed effects. First, the hydraulic gradient is at

an angle $\gamma(n)$, due to its pressure component. The angle $\gamma(n)$ can be obtained from Equation (28)

$$\frac{J_p}{J_n} = \tan(\alpha + \gamma(n)) = \tan(\alpha)\,\frac{e^{fN_f} - e^{fN_t}}{f(N_f - N_t)}\,e^{-fn} \qquad (36)$$

For depths smaller than N_{eq}, $N_t \leq n < N_{eq}$, $\gamma(n)$ is positive and the hydraulic gradient is deflected downslope, but for depths greater than N_{eq}, $N_{eq} < n \leq N_f$, $\gamma(n)$ is negative and the gradient is directed into the slope.

Comparison of Equations (34) and (36) shows that besides the deflection due to the hydraulic gradient, soil anisotropy is a second cause by which flow may be deflected in the saturated zone. Flow is deflected from the direction of the hydraulic gradient towards the direction of higher hydraulic conductivity, that is in the downslope direction. Depending on the values of f, N_f, N_t and a_r, the saturated flow can take different directions. For a given depth n, $\beta_s(n)$ increases with the anisotropy ratio, with the depth of the wetting front and with the thickness of the perched saturation zone. The angle of flow is never larger than 90 - a (i.e. parallel to the surface), but for small values of a_r, it can be negative for some values of n (flow is directed into the slope). For $a_r = 1$, that is, for an isotropic soil, flow is directed into the slope for all depths greater than N_{eq}.

A1.4.2 Vertical and horizontal components of flow

The vertical and horizontal components of flow are obtained by projecting the flow vector **q** on the z and x axes. Since there is no variation of front normal depth in the p direction, the values of N_f and N_t are constant for all the hillslope, and therefore independent from vertical depth z.

$$q_z(n) = q_n \cos(\alpha) + q_p \sin(\alpha) =$$

$$\left[K_{0_n} \frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \cos(\alpha) \right] \cos(\alpha) + [K_{0_p}\, e^{-fn} \sin(\alpha)] \sin(\alpha) \tag{37a}$$

$$q_x(n) = -\, q_n \sin(\alpha) + q_p \cos(\alpha) =$$

$$-\left[K_{0_n} \frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \cos(\alpha) \right] \sin(\alpha) + [K_{0_p}\, e^{-fn} \sin(\alpha)] \cos(\alpha) \tag{37b}$$

Upon simplification of Equations (37a) and (37b),

$$q_z(z) = K_{0_n} \left[a_r\, e^{-fz\cos(\alpha)} \sin^2(\alpha) + \frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \cos^2(\alpha) \right] \tag{38a}$$

$$q_x(z) = K_{0_n} \sin(\alpha) \cos(\alpha) \left[a_r\, e^{-fz\cos(\alpha)} - \frac{f\,(N_f - N_t)}{e^{fN_f} - e^{fN_t}} \right] \tag{38b}$$

As a consequence of flow deflection, vertical infiltration in the saturated zone is not constant with depth. It is the sum of a constant term and a term which decays exponentially with depth. The resulting horizontal

flow may be positive or negative (directed in the downslope or upslope direction), depending on the relative values of a_r, f, N_f and N_t. Its component directed into the slope is constant with vertical depth z, but its component directed out of the slope decreases exponentially with depth.

A1.4.3 Wetting and top front evolution in saturated infiltration

As in the case of unsaturated infiltration, detailed in the Appendix (Section A1.9), the equation of evolution of N_f is obtained through integration of the continuity equation upon substitution of the kinematic flow equation. The domain of integration Ω is now defined by $N_t < n_1 < N_f < n_2$. Normal flow q_n is given by Equation (29) at the level of n_1 (saturated zone) and by $R_i \cos(\alpha)$ at the level of n_2 (initial recharge rate). Integration of the two-dimensional continuity Equation (A1) over Ω gives,

$$\frac{dN_f}{dt} = \frac{K_{0_n} \dfrac{f(N_f - N_t)}{e^{fN_f} - e^{fN_t}} - R_i}{\theta_s - \theta(R_i, N_f)} \cos(\alpha) \qquad (39)$$

Derivation of the equation of evolution for N_t is entirely analogous to that for N_f. The domain of integration is now delimited by n_1 and n_2 defined by $n_1 < N_t < n_2 < N_f$. Normal flow is given by Equation(13) at depth n_1 and by Equation(29) at depth n_2. Top front evolution is given by

$$\frac{dN_t}{dt} = \frac{K_{0_n} \dfrac{f(N_f - N_t)}{e^{fN_f} - e^{fN_t}} - R}{\theta_s - \theta(R, N_t)} \cos(\alpha) \qquad (40)$$

Since we have R cos(α) > q_n in the saturated zone, the time derivative of N_t is negative, i.e., N_t approaches the terrain surface. Eventually, N_t will reach the surface (N_t = 0), and from then on we must have

$$N_t = 0; \qquad \frac{dN_t}{dt} = 0 \tag{41}$$

A1.5 The infiltration model

For N_t = 0, compatibility of (41) with (40) requires that R = q_n in the saturated zone, with R now representing the rate of infiltration. The rate of infiltration is now smaller than the rainfall rate and infiltration excess runoff is generated. This situation also applies to the case R > K_{0n}. If rainfall rate is higher than the saturated hydraulic conductivity at the surface, there is no initial unsaturated infiltration. Infiltration occurs under saturated conditions from the beginning. The top front is permanently at the surface and wetting front evolution is only governed by Equation (39).

In summary, if we designate the rate of infiltration by R_{inf} and the rate of runoff by R_r,

$$R_{inf} = \begin{cases} N_t > 0 : & R\cos(a) \\ N_t = 0 : & K_{0n}\dfrac{f N_f}{(e^{f N_f} - 1)}\cos(\alpha) \end{cases} \tag{42}$$

$$R_r = R - R_{inf} \tag{43}$$

When the top front reaches the surface of the soil, infiltration into the soil is no longer controlled by the infiltration capacity of the upper layer (surface normal hydraulic conductivity K_{0n}), but by the infiltration capacity of all the saturated profile, equal to the equivalent hydraulic conductivity K_{eq} (harmonic mean of the hydraulic conductivities of the saturated zone). Since for a constant rainfall rate R the saturated area is expanding, the infiltration capacity of the saturated area decreases in time.

A1.6 Lateral subsurface discharge from a vertical cross-section

In this Section we obtain an expression for the lateral discharge from a vertical cross-section of the hillslope soil. The total discharge from the wetted soil, represented by Q, from a vertical cross-section of width W is obtained through integration over the wetted depth (i.e. from 0 to $\frac{N_f}{\cos(a)}$) of the horizontal component of flow,

$$Q = W \int_0^{\frac{N_f}{\cos(\alpha)}} q_x(z)\, dz \qquad (44)$$

Replacing this integral by its components within the unsaturated and saturated zones,

$$Q = W \cdot \left[\int_0^{\frac{N_t}{\cos(\alpha)}} q_x(z)\, dz + \int_{\frac{N_t}{\cos(\alpha)}}^{\frac{N_f}{\cos(\alpha)}} q_x(z)\, dz \right] \qquad (45)$$

Substituting Equations (20b) and (38b) in the first and second integrals of Equation (45), respectively, we obtain

$$Q = W \sin(\alpha) \left\{ [N_t R (a_r - 1)] + \left[K_{0_n} \frac{a_r}{f} (e^{-fN_t} - e^{-fN_f}) \right] - \left[K_{0_n} \frac{f (N_f - N_t)^2}{e^{fN_f} - e^{fN_t}} \right] \right\} \quad (46)$$

For a rainfall rate lower than the initial infiltration capacity, i.e. for $R < K_{0n}$, lateral discharge is given by the first square-bracketed term of (46) while we have $N_t = N_f < N^*(R)$. In this case, lateral discharge increases with the anisotropy ratio. After perched saturation has developed, i.e. for $N_t < N^*(R) < N_f$, lateral discharge is given by all three terms of Equation (46) while we have $N_t > 0$. In the advanced stages of infiltration, we have $N_t = 0$ and N_f very large. Therefore, the first term in Equation (46) equals zero, the third approaches zero and the second term has a limit value for large N_f, given by,

$$\lim_{t \to \infty} Q = W \sin(\alpha) K_{0_n} \frac{a_r}{f} \quad (47)$$

The limit value increases with the anisotropy ratio and the surface saturated hydraulic conductivity and decreases with the rate of conductivity decay with depth, f.

A1.7 Infiltration model sensitivities

We present model results for various soil parameter values, rainfall rates, and terrain inclinations. A soil with the following basic characteristics was selected to obtain numerical results:

K_{0n} : 20 mm h^{-1}

θ_s : 0.5

θ_r : 0.05

ε : 4.0

The soil is assumed to be almost dry at the beginning of the event. An initial recharge rate of R_i = 0.01 mm h^{-1} defines this initial moisture distribution.

***Table 1**: Numerical values used in the simulations*

Parameter				Reference values		
Rainfall rates (mm h^{-1})	2	5	7	10		
Slope angle (°)		0	10	20	45	
Anisotropy ratio		1	5	10	100	
Parameter f (m^{-1})			10	5	1	0.1

Sensitivity analyses were carried out for the rest of the variables that affect the results, namely: rainfall rate (R), terrain slope angle (α), anisotropy ratio (a_r) and decrease of permeability with depth (parameter f). Table 1 shows the numerical values adopted for the variables in the sensitivity analysis. A basic set of values (central column) was adopted as a reference. Sensitivity analyses were performed changing only one variable at a time, setting the rest to their basic reference values. The

model basically involves the numerical integration of the evolution equations (21), (39), and (40) for a period equal to 60 hours.

Figures (7) to (9) show the position of both the wetting front and the top of the zone of perched saturation for a variety of rain rates, f-values and slopes. The top front is shown to often diverge from the bottom front, indicating the development and growth of a zone of perched saturation. Figure (7) shows the evolution in time of front depth for different rainfall intensities. It can be seen that except for the case with lowest rainfall intensity, saturation is reached at a relatively shallow position. Figure (8) shows the importance of the parameter f for the definition of the depth N*, and consequently in the position of the fronts and evolution of the saturated area. Figure (9) shows that the sensitivity to the slope angle α is small except for steep inclinations. There is no sensitivity analysis to the

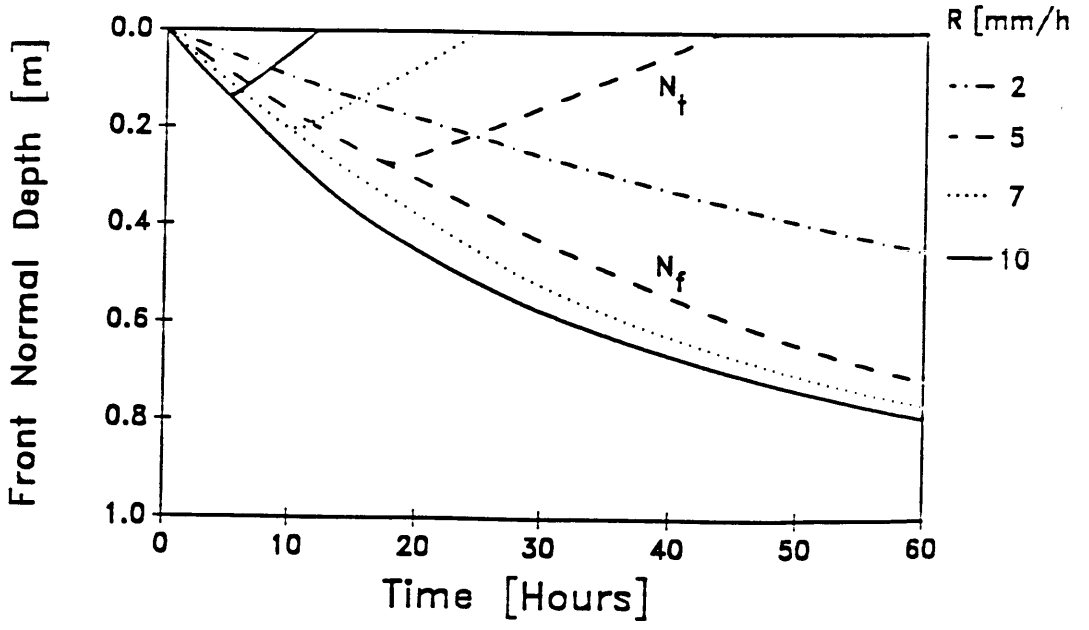


***Figure 7**: Front position as a function of time for different rainfall intensities. The slope angle is 20°. Normal hydraulic conductivity at the surface is 20 $mm.h^{-1}$, with an exponential decay rate of 0.005 mm^{-1}. The soil is anisotropic, with an anisotropy ratio of 10.*

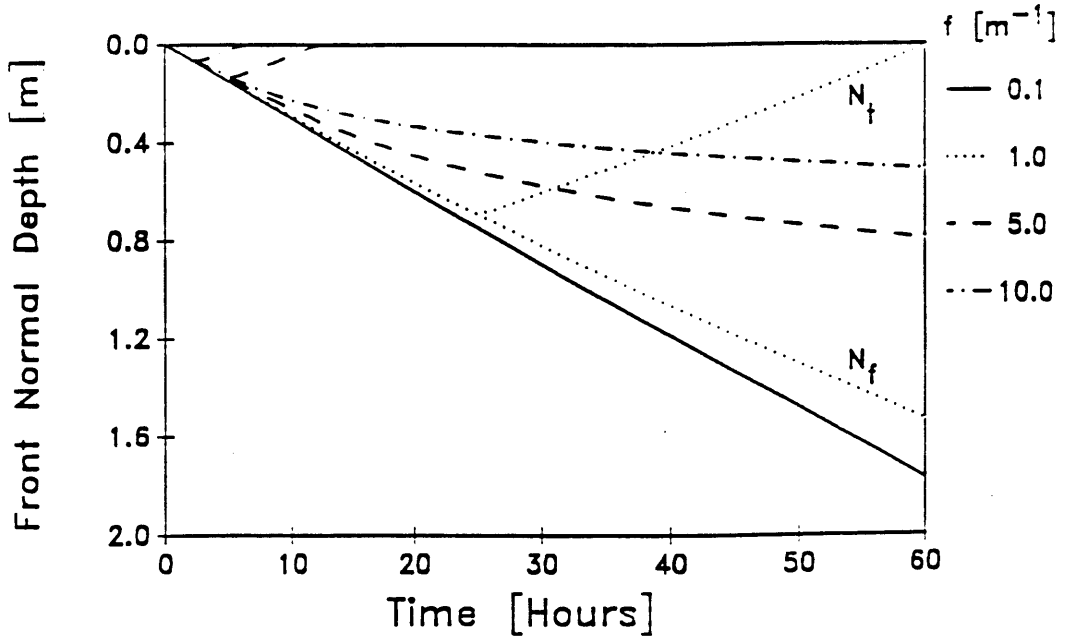


Figure 8: *Front position as a function of time for different values of the exponential decay rate of hydraulic conductivity (parameter f). Normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the rainfall intensity is 10 mm.h^{-1}, the slope angle is 20° and the anisotropy ratio is 10.*

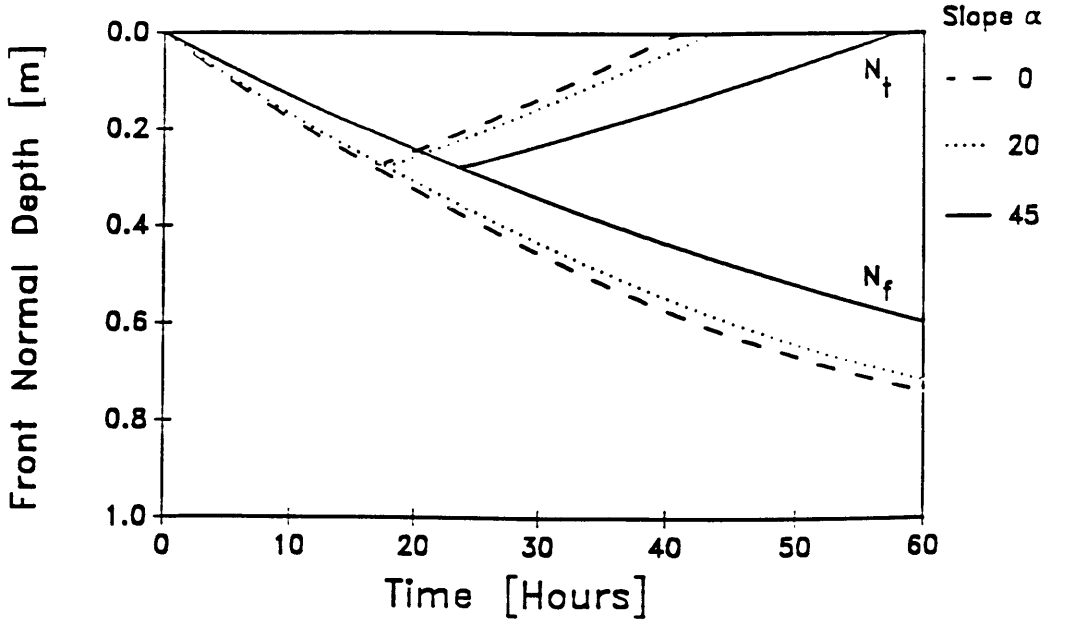


Figure 9: *Front position as a function of time for different slope angles. Rainfall intensity is 5 mm.h^{-1}, normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the anisotropy ratio is 10.*

anisotropy ratio because front evolution (and therefore front position) is independent of soil anisotropy parallel to the soil surface.

Figures (10) to (12) show the effect of model parameters in the generation of hillslope runoff. All rainfall intensities used in Figure (10) are smaller than the surface normal saturated conductivity (20 mm h^{-1}); three of them generate hillslope runoff during the first 60 hours of storm and in a relatively dry soil. At the beginning of the event, the infiltration rate is equal to rainfall intensity because the process is controlled by the infiltration capacity of the unsaturated upper layers of the soil. When the top front eventually reaches the surface, infiltration is controlled by the equivalent permeability of the saturated area, which is an average of the normal conductivities of the saturated zone. Average conductivity is smaller than that of the upper layers of the soil and smaller than rainfall

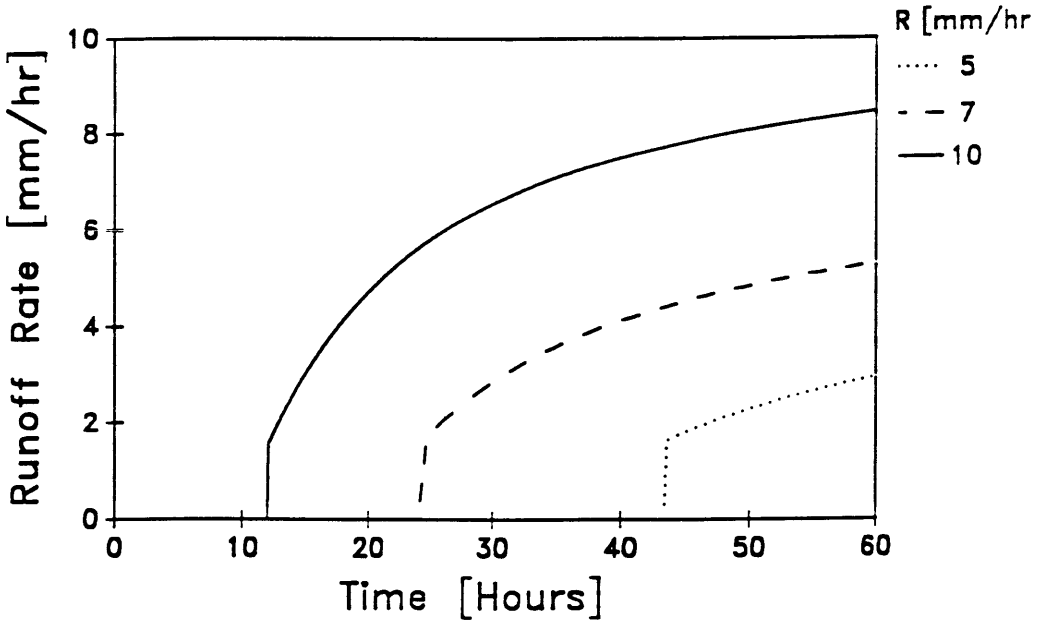


***Figure 10**: Hillslope runoff generation as a function of time for different rainfall intensities. The slope angle is 20°, normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the soil is anisotropic, with an anisotropy ratio of 10.*

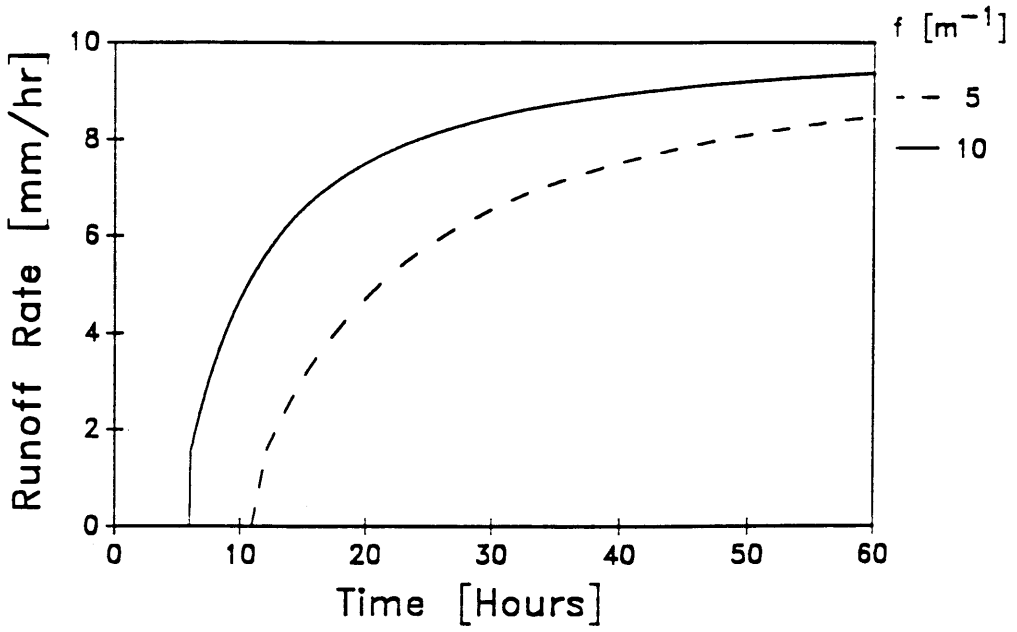


Figure 11: *Hillslope runoff generation as a function of time for different vales of the exponential decay rate of hydraulic conductivity (parameter f). Normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the rainfall intensity is 10 mm.h^{-1}, the slope angle is 20° and the anisotropy ratio is 10.*

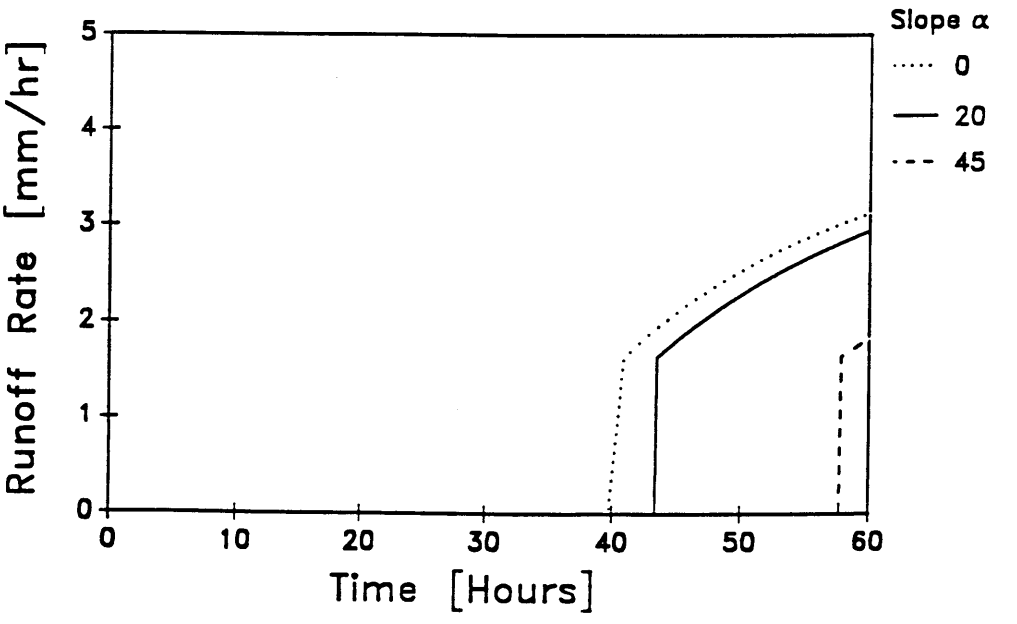


Figure 12: *Hillslope runoff generation as a function of time for different slope angles. Rainfall intensity is 5 mm.h^{-1}, normal hydraulic conductivity at the surface is 20 mm.h^{-1}, then exponential decay rate is 0.005 mm^{-1} and the anisotropy ratio is 10.*

intensity. That produces a sharp decrease in the infiltration capacity of the soil and the consequent increase in the runoff. From that point on, runoff generation is controlled by the equivalent permeability of the saturated area, which decreases as the wetting front advances further into the soil. Figures (11) and (12) show the sensitivity of runoff generation to the conductivity parameter f and to the slope angle, with similar effects to those already commented for the front evolution.

Figures (13) to (18) show streamlines at different stages of front evolution for soils with anisotropy ratios of 1 and 10. Figures (13) and (14) correspond to front position after 5 hours of rainfall. Most of the wetted area is unsaturated and, therefore streamlines are vertical lines for the case $a_r = 1$ and at an angle $\beta_u = \tan^{-1}[10 \tan(10°)] - 10°$ for the case $a_r = 10$. Figures (15) and (16) show the streamlines after 10 hours of rainfall, when the top front is almost at the surface and most of the wetted area is saturated. Since the saturated area is still relatively shallow the curvature of the streamlines is not large, specially for the case $a_r = 1$. Figures (17) and (18) show the streamlines after 60 hours of rainfall, with the entire wetted profile (up to the wetting front) saturated. As it can be seen in Figure (17) (isotropic case), soil heterogeneity translates into an overall downslope deviation of flow, although flow in the lower part of the saturated area is directed into the slope. Figure (18) illustrates how anisotropy affects the direction of flow. After many hours of rainfall, flow in the upper part of the saturated area is almost parallel to the surface, and therefore parallel flow dominates over vertical flow. These results are consistent with field observations of subsurface storm runoff generation [*Tanaka et al.*,1987], that link storm runoff mostly to the movement of shallow groundwater parallel to the surface. This phenomenon is usually

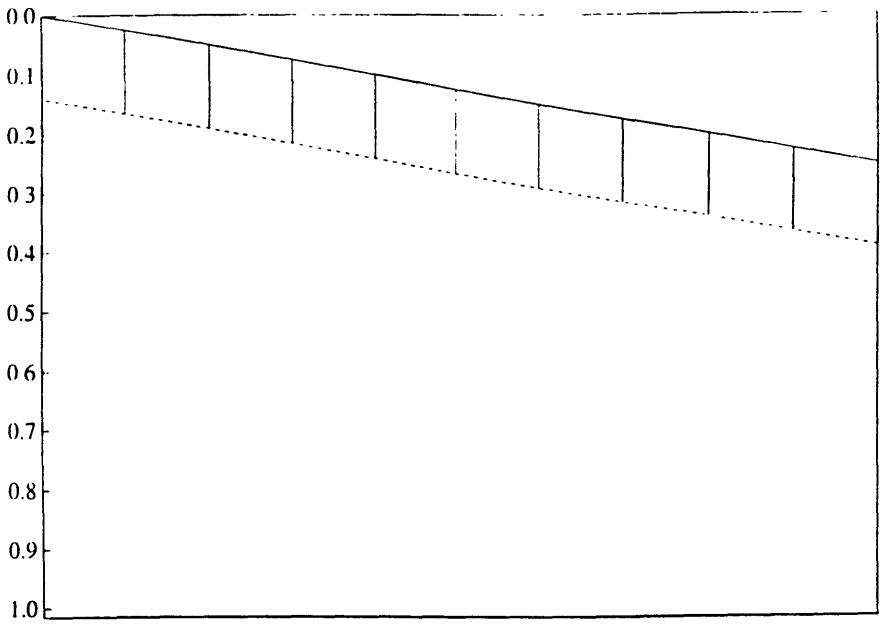


Figure 13: *Representation of flow lines after 5 hours of infiltration for an isotropic soil with slope angle of 10°. Normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

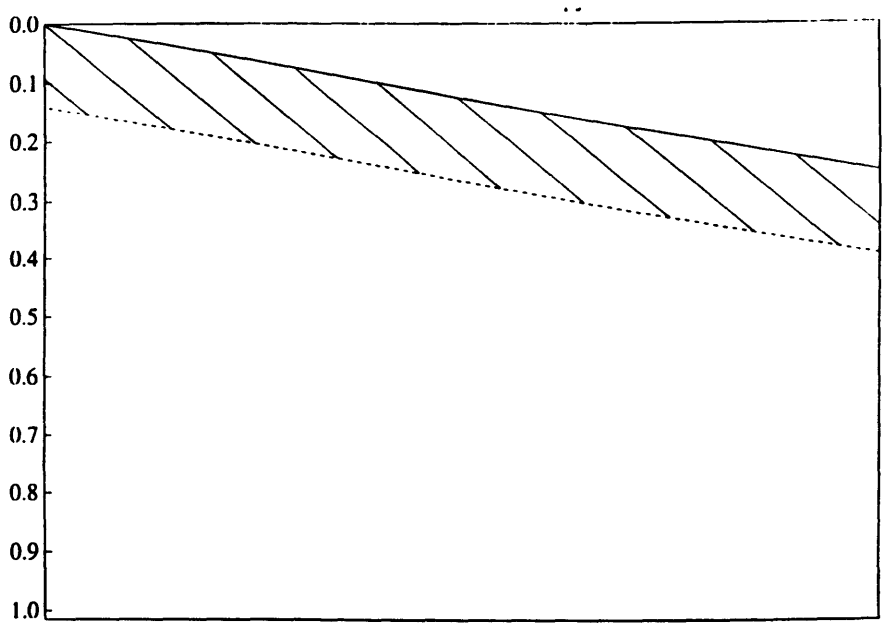


Figure 14: *Representation of flow lines after 5 hours of infiltration for a soil with slope angle of 10° and anisotropy ratio of 10. The normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

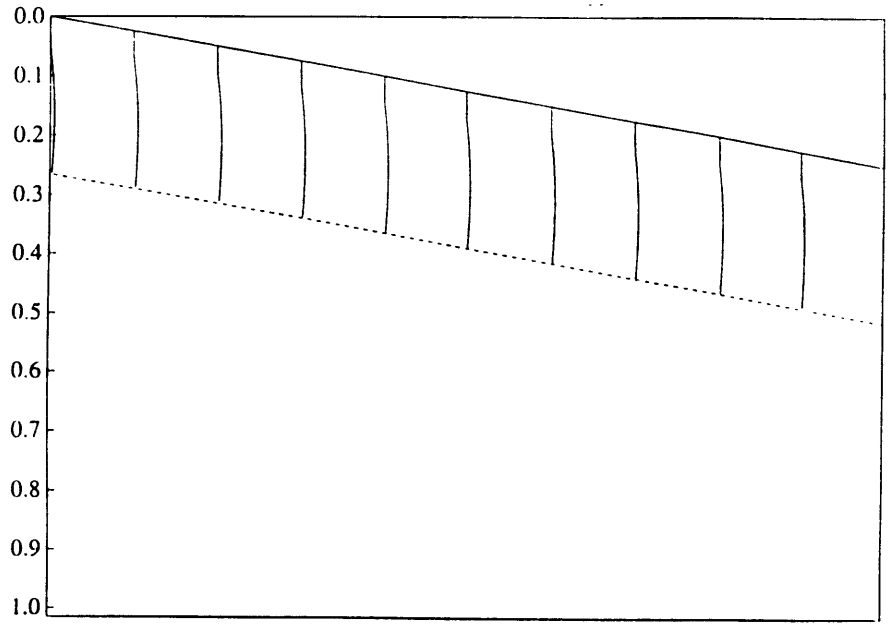


Figure 15: *Representation of flow lines after 10 hours of infiltration for an isotropic soil with slope angle of 10°. The normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

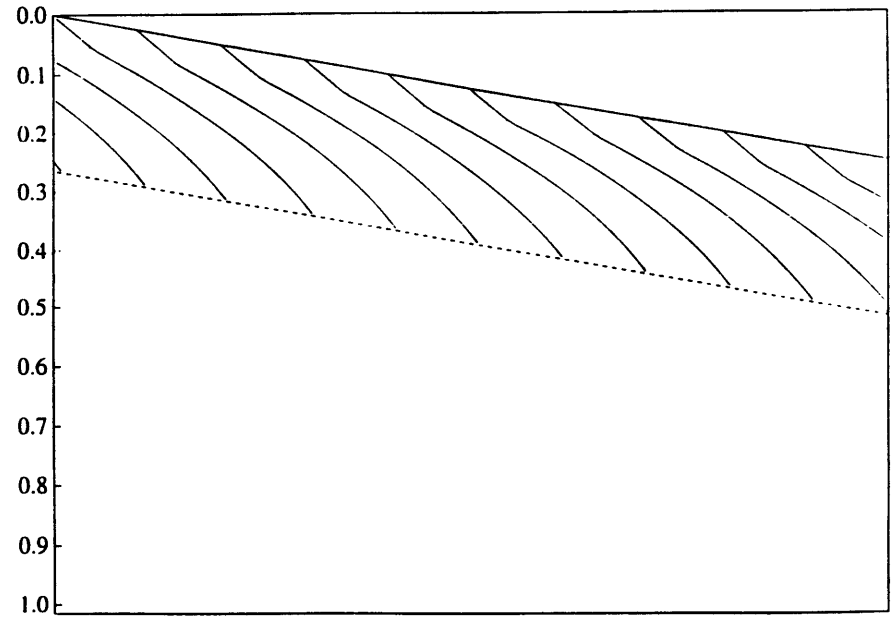


Figure 16: *Representation of flow lines after 10 hours of infiltration for a soil with slope angle of 10° and anisotropy ratio of 10. The normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

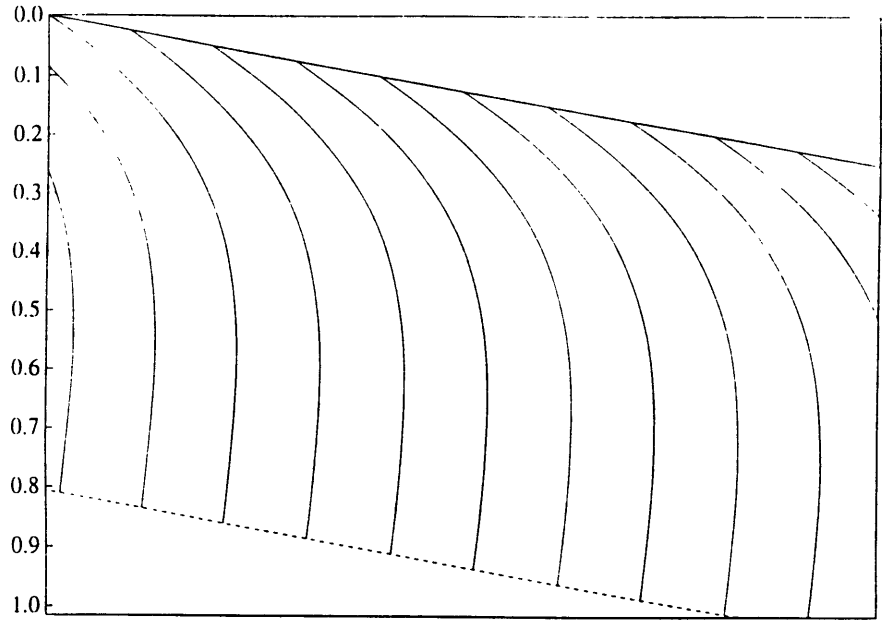


Figure 17: *Representation of flow lines after 60 hours of infiltration for an isotropic soil with slope angle of 10°. The normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

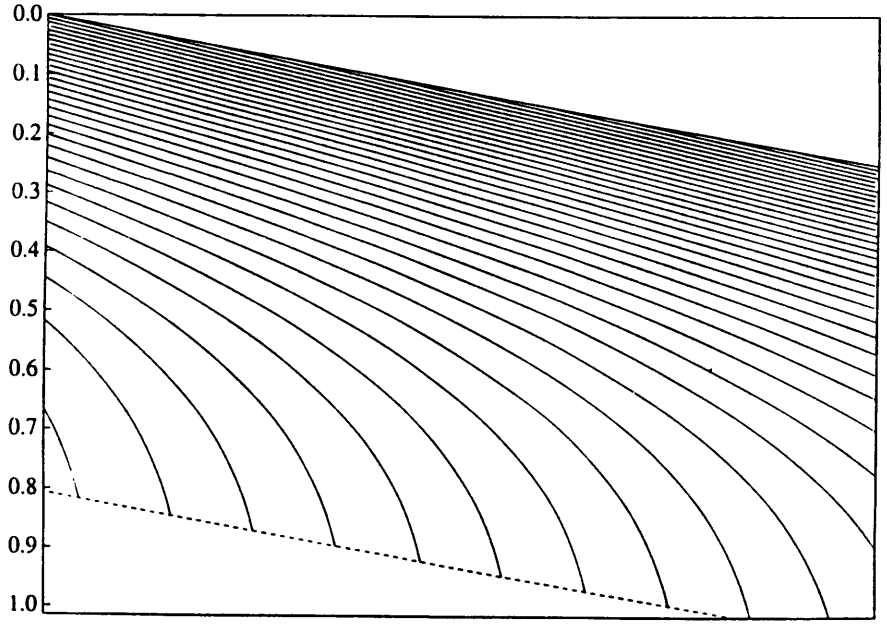


Figure 18: *Representation of flow lines after 60 hours of infiltration for a soil with slope angle of 10° and anisotropy ratio of 10. the normal hydraulic conductivity at the surface is 20 mm.h^{-1}, the exponential decay rate is 0.005 mm^{-1} and the rainfall intensity is 10 mm.h^{-1}.*

attributed in the literature to macropore flow, but it can be to some extent represented by Darcian matrix flow with a high anisotropy ratio.

A1.8 Conclusions

A one-dimensional model of infiltration and subsurface lateral flows that considers the effects of anisotropy and vertical heterogeneity has been presented. The model is based on the kinematic approximation and on a simplified mathematical description of soil anisotropy and heterogeneity which allow for analytical treatment of the problem. Outputs of the model are the wetting front advance, the evolution of a perched saturation zone, and the generation of lateral unsaturated and saturated subsurface flow.

The kinematic approximation leads to the formation of a sharp wetting front. With hydraulic conductivity decreasing with depth, prolonged infiltration leads to the development of saturation at a depth in the soil profile where infiltration capacity equals the percolation rate. As the wetting front progresses further below this critical depth, discharge is limited by the lower conductivity at the wetting front and moisture progressively accumulates within a zone of perched saturation. Evolution equations for the location of the wetting front and the growth of the zone of perched saturation have been derived.

The effect of soil heterogeneity and anisotropy on infiltration is flow deflection in the downslope direction. Flow deviation with respect to the vertical has been analyzed for both unsaturated and saturated regimes. Numerical results have been presented to illustrate how this one-

dimensional model can reproduce some of the observed two-dimensional effects of subsurface stormflow generation in sloped soils .

Since simple analytical expressions for front advance are derived, the one-dimensional infiltration model presented here is computationally efficient. It can be used as the basis for estimating local runoff generation in a distributed basin model. The expression for horizontal flow in a vertical cross section allows for horizontal flow transfer between the distributed elements. Further reports will focus on the adaptation of the model to variable rainfall rates and to address the problem of the variability in the parallel direction.

A1.9 Appendix to the paper

In this Appendix the equation for the time evolution of the wetting front is derived. The continuity equation in the *(n,p)* coordinate system is

$$\frac{\partial \theta}{\partial t} + \frac{\partial q_n}{\partial n} + \frac{\partial q_p}{\partial p} = 0 \qquad \text{(A1)}$$

Equation (A1) can be integrated in the domain W that includes the wetting front (see Figure A1).Domain Ω is delineated by the planes $p=p_1$, $p=p_2$, $n=n_1$ and $n=n_2$, with $n_1 < N_f < n_2$. Integration of the continuity equation in the domain Ω gives

$$\int_\Omega \left(\frac{\partial \theta}{\partial t} + \frac{\partial q_n}{\partial n} + \frac{\partial q_p}{\partial p} \right) d\Omega = \int_\Omega \left(\frac{\partial \theta}{\partial t} \right) d\Omega + \int_\Omega \left(\frac{\partial q_n}{\partial n} + \frac{\partial q_p}{\partial p} \right) d\Omega = 0 \qquad \text{(A2)}$$

Interchanging the integral and the derivative in the first term and applying Green's theorem to the second term gives

$$\frac{d}{dt}\left(\int_{\Omega}\theta\, d\Omega\right) + \int_{\partial\Omega}(\vec{q}\,.\,\vec{i}_N)\, d\partial\Omega = 0 \qquad (A3)$$

where $\mathbf{i_N}$ is a unit vector normal to the boundary of Ω, $d\Omega$. The time rate of change in moisture content within Ω is balanced by the flux of $\mathbf{q}$ across its encompassing boundary $d\Omega$.

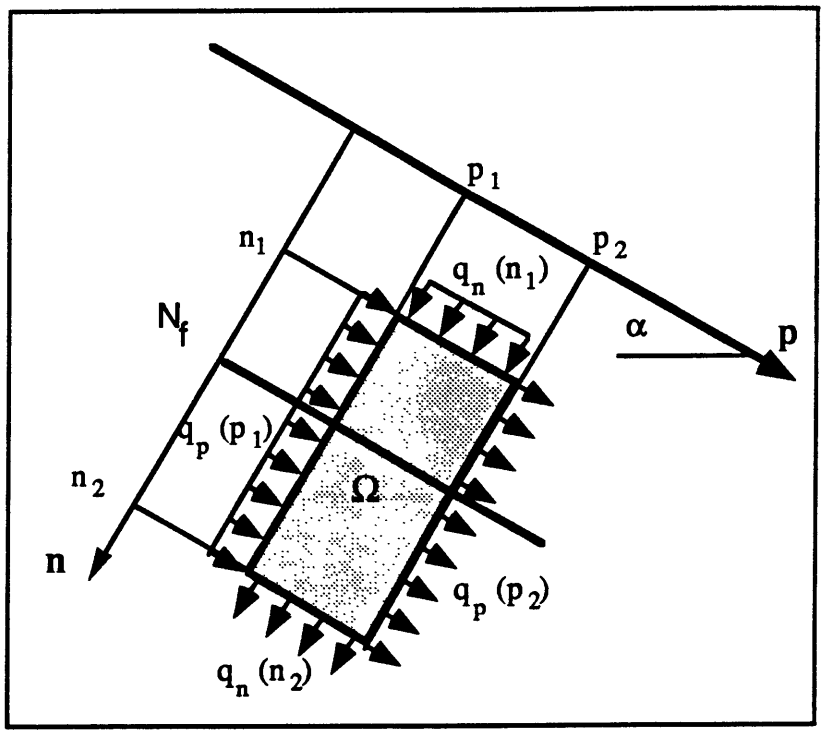


***Figure A1:** Integration domain Ω for the continuity equation. The wetting front Nf is between n1 and n2.*

The soil moisture profile is given by Equation (15). The rainfall rate is R above the wetting front ($n_1<n<N_f$). Prior to the initiation of rainfall the soil moisture profile can be described by a flow rate $R_i < R$. Therefore, the initial moisture profile is given by (15) upon substitution of R by R_i for

depths below N_f ($N_f<n<n_2$), a region still unaffected by the moisture wave. Upon substitution of the soil moisture profile in the integral of the first term of Equation (A3),

$$\int_\Omega \theta \, d\Omega = \int_{p_1}^{p_2}\int_{n_1}^{n_2} \theta(n)\, dn\, dp = (p_2 - p_1)\left[\int_{n_1}^{N_f} \theta(n)\, dn + \int_{N_f}^{n_2} \theta(n)\, dn\right] =$$

$$(p_2 - p_1)\left[\left(\frac{R}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r)\left(\frac{\varepsilon}{f}\right)\left(e^{\frac{f}{\varepsilon}N_f} - e^{\frac{f}{\varepsilon}n_1}\right) + \theta_r(N_f - n_1) + \right.$$

$$\left. \left(\frac{R_i}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r)\left(\frac{\varepsilon}{f}\right)\left(e^{\frac{f}{\varepsilon}n_2} - e^{\frac{f}{\varepsilon}N_f}\right) + \theta_r(n_2 - N_f)\right] \qquad \text{(A4)}$$

Differentiation of (A4) with respect to time yields the first term in Equation (A3),

$$\frac{d}{dt}\left(\int_\Omega \theta \, d\Omega\right) = (p_2 - p_1)\left\{\left[\left(\frac{R}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r)\, e^{\frac{f}{\varepsilon}N_f} + \theta_r\right] - \right.$$

$$\left. \left[\left(\frac{R_i}{K_{0_n}}\right)^{\frac{1}{\varepsilon}} (\theta_s - \theta_r)\, e^{\frac{f}{\varepsilon}N_f} + \theta_r\right]\right\} \frac{dN_f}{dt} \qquad \text{(A5)}$$

Recalling the expression for $\theta(R,n)$ given by Equation (15),

$$\frac{d}{dt}\left(\int_\Omega \theta \, d\Omega\right) = (p_2 - p_1)[\theta\,(R\,,\, N_f) - \theta\,(R_i\,,\, N_f)]\frac{dN_f}{dt} \qquad \text{(A6)}$$

The second term of Equation (A3) is the flux of the vector **q** across the boundary of Ω, which leads to

$$\int_{\partial\Omega} (q \cdot i_N)\, d\partial\Omega = -\int_{n_1}^{n_2} q_p|_{p=p_1}\, dn - \int_{p_1}^{p_2} q_n|_{n=n_1}\, dp + \int_{n_1}^{n_2} q_p|_{p=p_2}\, dn + \int_{p_1}^{p_2} q_n|_{n=n_2}\, dp \quad (A7)$$

Since the rate of flow parallel to the hillslope is constant with depth

$$q_p|_{p=p_1} = q_p|_{p=p_2} \quad \forall n$$

and

$$\int_{\partial\Omega} (q \cdot i_N)\, d\partial\Omega = -\int_{p_1}^{p_2} q_n|_{n=n_1}\, dp + \int_{p_1}^{p_2} q_n|_{n=n_2}\, dp \quad (A8)$$

For all depths above the wetting front ($n < N_f$), q_n is a function of R, given by Equation (13). Below the wetting front ($n>N_f$) q_n is also given by Equation (13), substituting R by R_i. It then follows that the terms of Equation (A8) are

$$-\int_{p_1}^{p_2} q_n|_{n=n_1}\, dp = -\int_{p_1}^{p_2} R\cos(\alpha)\, dp = -R\cos(\alpha)\,(p_2 - p_1)$$

$$\int_{p_1}^{p_2} q_n|_{n=n_2}\, dp = \int_{p_1}^{p_2} R_i\cos(\alpha)\, dp = R_i\cos(\alpha)\,(p_2 - p_1)$$

Using Equations (A6), (A7), and (A8) in (A3) we obtain the rate of movement of the wetting front,

$$\frac{dN_f}{dt} = \frac{(R - R_i)\cos(\alpha)}{\theta(R, N_f) - \theta(R_i, N_f)} \quad (A9)$$

The rate of advance of the wetting front depends on the difference between the rainfall rate and the initial recharge ($R - R_i$) and the difference between the moisture distributions corresponding to R and R_i at the depth of the wetting front.

A1.10 Acknowledgements

This research was partially supported by the Arno Project of the National Research Council of Italy, through a cooperative agreement with the University of Florence, Italy. Other sponsors included the U.S. Army Research Office (Grant DAALO-3-89-K-0151), the National Science Foundation (Grant CES-8815725) and the National Weather Service (cooperative agreement NA86AA-D-HY123). Luis Garrote was sponsored by the Spanish Ministry of Education and Science. The contributions of David Tarboton to initial model development are gratefully acknowledged.

A1.11 References

Beven, K.J. (1981), Kinematic subsurface stormflow, *Water Resources Research*, vol.17, n.5, pp 1419-1424.

Beven, K. (1982), On subsurface stormflow: an analysis of response times, *Hydrological Sciences Journal*, vol.4, pp 505-521.

Beven, K. (1984), Infiltration into a class of vertically non-uniform soils, *Hydrological Sciences Journal*, vol.29, pp 425-434.

Brooks, R.H., and A. T. Corey (1964), Hydraulic properties of porous media, Hydrology Paper 3, Colorado State University, Fort Collins, Colo., 1964.

Charbeneau, R.J. (1984), Kinematic models for soil moisture and solute transport, *Water Resources Research*, vol.20, n.6, pp 699-706.

Clapp, R.B. and G.M. Hornberger (1978), Empirical equations for some soil hydraulic properties, *Water Resources Research*, vol.14, n.4, pp 601-604.

Dagan,G. and E. Bressler (1983), Unsaturated flow in spatially variable fields. 1.- Derivation of models of infiltration and redistribution, *Water Resources Research*, vol.19, n.2, pp 413-420.

McCord, J.T., and D.B. Stephens (1987), Lateral moisture flow beneath a sandy hillslope without an apparent impending layer, *Hydrological Processes Journal,* vol 1, pp 225-238.

Mualem, Y. (1978), Hydraulic conductivity of unsaturated porous media: generalized macroscopic approach, *Water Resources Research*, vol.14, n.2, pp 325-334.

Philip, J.R. (1991), Hillslope infiltration: planar slopes, *Water Resources Research*, vol.27, n.1, pp 109-117.

Smith, R. E., and R. H. B. Hebbert (1983), Mathematical simulation of interdependent surface and subsurface hydrologic processes, *Water Resources Research*, vol.19, n.4, pp 987-1001.

Tanaka, T., M. Yashuhara, H. Sakai, and A. Marui (1988), The Hachioji experimental basin study - Storm runoff processes and the mechanisms of its generation, *Journal of Hydrology*, vol 102, pp 139-164.

Yeh, T.C.J., L.W. Gelhar, and A.L. Gutjahr (1985), Stochastic analysis of unsaturated flow in heterogeneous soils 1.- Statistically isotropic media *Water Resources Research*, vol.21, n.4, pp 447-456.

Zaslavsky, D., and A.S. Rogowski (1969), Hydrologic and Morphologic implications of anisotropy and infiltration in soil profile development, *Soil Sci. Soc. Amer. Proc*, vol.33, pp 594-599.

Zaslavsky, D., and G. Sinai (1981), Surface hydrology: IV -- Flow in sloping, layered soil, *Journal of the Hydraulics Division*, Proceedings ASCE, vol.107, n.HY1, pp 53-64.

APPENDIX 2

Data Description

This appendix contains format descriptions and naming conventions for the files used by the RIBS environment. The core of the external storage system is the hydrologic database, which contains distributed variables and hydrographs. There are also other auxiliary input and output files used by the different programs in the system.

A2.1 Naming conventions and file organization

Data are stored in files located in different directories in a *unix* system, and they are accessed through path and file name. There are two types of files: static and dynamic files. Static files correspond to variables which do not change during a simulation, and dynamic files correspond to variables which change over time, and therefore, have a time tag associated with them. The most general file name corresponds to the dynamic files and consists of three sections, separated by dot (.) symbols. The first section is designated '*root*', the second section is designated '*time tag*' and the third section is designated '*extension*'. A full filename is therefore '*pathroot.time tag.extension*'. Static files do not have a time tag section, and only consist of root and extension: '*pathroot.ext*.

The root section is used to identify different basins or different model runs for the same basin within a given directory. The time tag section contains the time tag of the file, in the format *hhhh:mm*. The time tag may have different interpretations, depending on the variable. If the variable stored in the file is an instantaneous value, the time tag represents the time at which the file was generated. If the variable is an average or cumulative value, the time tag is interpreted as the beginning or the end of the time interval associated with the variable. For instance, a rainfall file corresponding to cumulative rainfall between 16:00 and 17:30 hours of February 7th would have a time tag of 0904:30, even if the file is actually generated some time later (for instance, at 17:48 hours). The extension is used to differentiate between different variables of the same group.

Files are organized by the user within the directory tree. Six groups of files are considered, each one of which can be stored in a different directory. A model of file organization is shown in Figure A2.1, although other user-defined models are also valid. File groups are organized as follows:

- *Geomorphology directory*: Contains geometric and pedologic data about the basin. All data stored in this directory are permanent, and correspond to the basin, not to a single storm or model run. The directory contains raster descriptions of distributed variables and a file of special format: the soil types file, which contains parameter values for every soil type.
- *Auxiliary rainfall directory*. Contains data used by the rainfall acquisition module to emulate the arrival of rainfall information in time.

- *Measured rainfall directory*: Contains measured rainfall files. All files corresponding to the same storm share the same root and extension, and are differentiated by their time tags.

- *Forecasted rainfall directory*: Contains forecasted rainfall files. All files corresponding to the same model run share the same root and extension, and are differentiated by their time tags.

- *State variables directory*: Contains basin state variables. All files corresponding to the same run share the same root. There are four state variables, and each one has a different extension. Files corresponding to the same state variable are differentiated by their time tags.

- *Hydrographs directory*: Contains basin hydrographs. There are as many file groups as gauges are defined by the user. Each gauge has a different root name, but all share the same extension. Files corresponding to the same gauge are differentiated by their time tags.

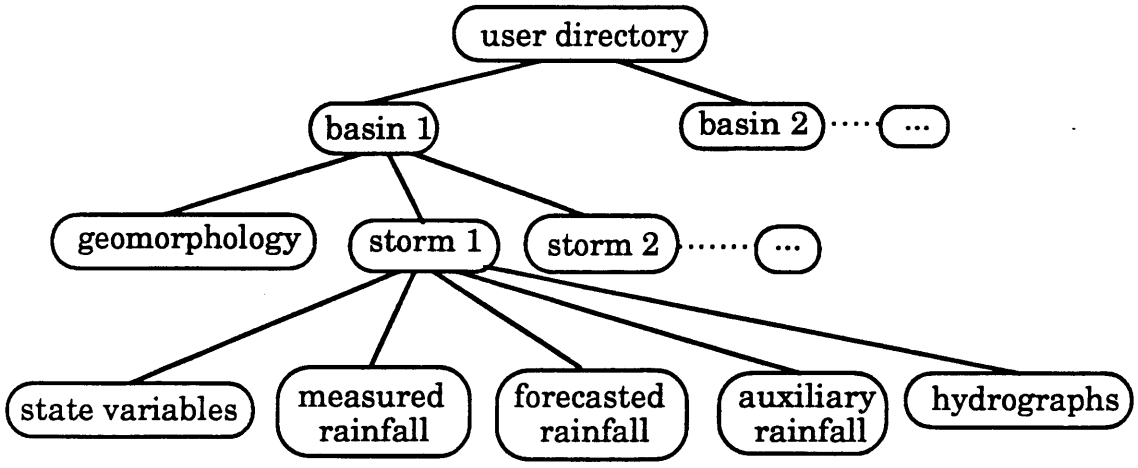


Figure A2.1 *Proposed generic structure of the directory tree to store files in the RIBS system*

The user can specify paths, roots and extensions. Time tags are automatically assigned by the system. File organization is therefore defined by the user within the constraints imposed by the system. The user can store all files in the same directory (by setting all directories to the same path) or distribute them in different directories. Root names may correspond to a basin, to a storm or to a particular model run with a given parameter set. A sample file organization, corresponding to the storm of November 1991 for the Sieve case study, is shown in Figure A2.2.

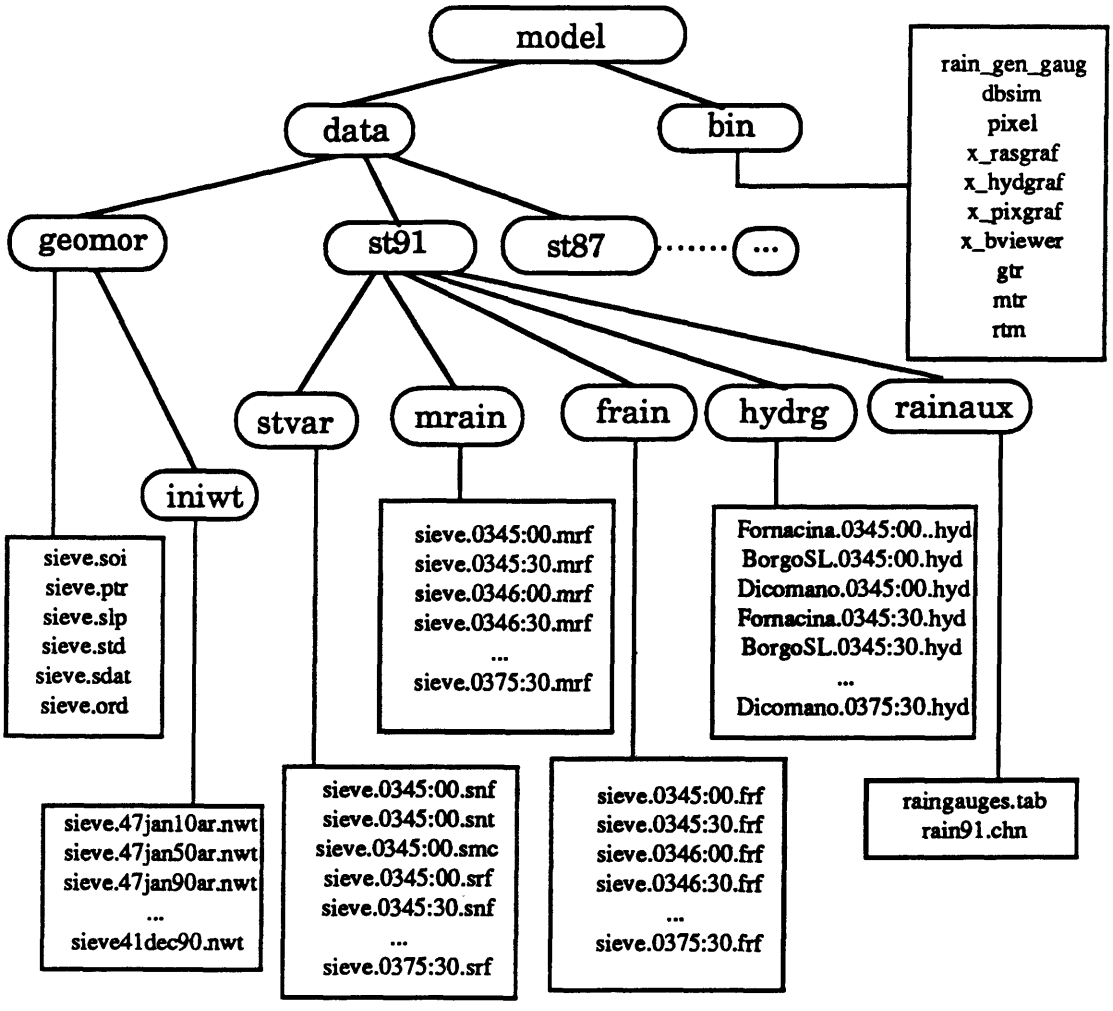


***Figure A2.2** Sample file organization for the case study of the November 1991 storm in the Sieve basin*

A2.2 File format

The RIBS system deals with several file types. This section specifies the file format for every file type. Files can be classified in three groups. The first group consists of the dynamic database, which contains model data and results. The second group consists of trace files, and contains secondary output from the model, not included in the model user interface. The third group consists of auxiliary files, which contain information needed by the model, but whose format cannot be adapted to the standards of the database. Formats of file types in every group are defined as follows.

A2.2.1 Database files

The database stores information in a format accessible by the user interface. It is composed of RIBS objects which can be stored into files by the system. Three types of objects have a permanent representation in the RIBS environment: *Raster*, *Hydrograph* and *Pixel*. Files representing the spatial distribution of a variable are stored in the *Raster* format, files representing the temporal distribution of one or more variables, generally hydrographs, are stored in the *Hydrograph* format and files representing the state of a pixel are stored in the *Pixel* format.

Raster format

The raster format is used to store spatial information in an efficient way. The raster format adopted by RIBS follows the specifications

developed by Becchi et al, (1991) for distributed data structures in the Arno project. What follows here is a summary of the standard format.

The data stored in the raster are distributed spatially in the nodes of a rectangular grid. In order to optimize space storage for sparse grids, only the nodes which contain data values are stored in the raster. The nodes within the boundaries of the basin are grouped in rows. Each row is composed of one or more clusters, which are groups of contiguous non-empty nodes. Data are stored into and retrieved from the raster through a row and cluster indexing system, so that there is a one-to-one correspondence between position in the raster and coordinates in the basin. Figure A2.3 illustrates the raster respresentation of the basin.

Within the raster, data are stored in binary format, assigning only four significant digits to every quantity. That resolution is usually enough, considering that raster files store the spatial distribution of physical variables, and it is unlikely that the precision of the measurement or estimation be higher than four orders of magnitude. In order to facilitate the storage, data are transformed according to the equations

$$R = (V - a)\, 10^{-m} \qquad \text{(A2.1a)}$$

$$V = a + R10^{m} \qquad \text{(A2.1b)}$$

where V is the real value, R is the value stored in the raster and a, m are the transformation parameters. a and m are defined so that the range of values to store can be covered by the range 0 - 32767 offered by the binary format.

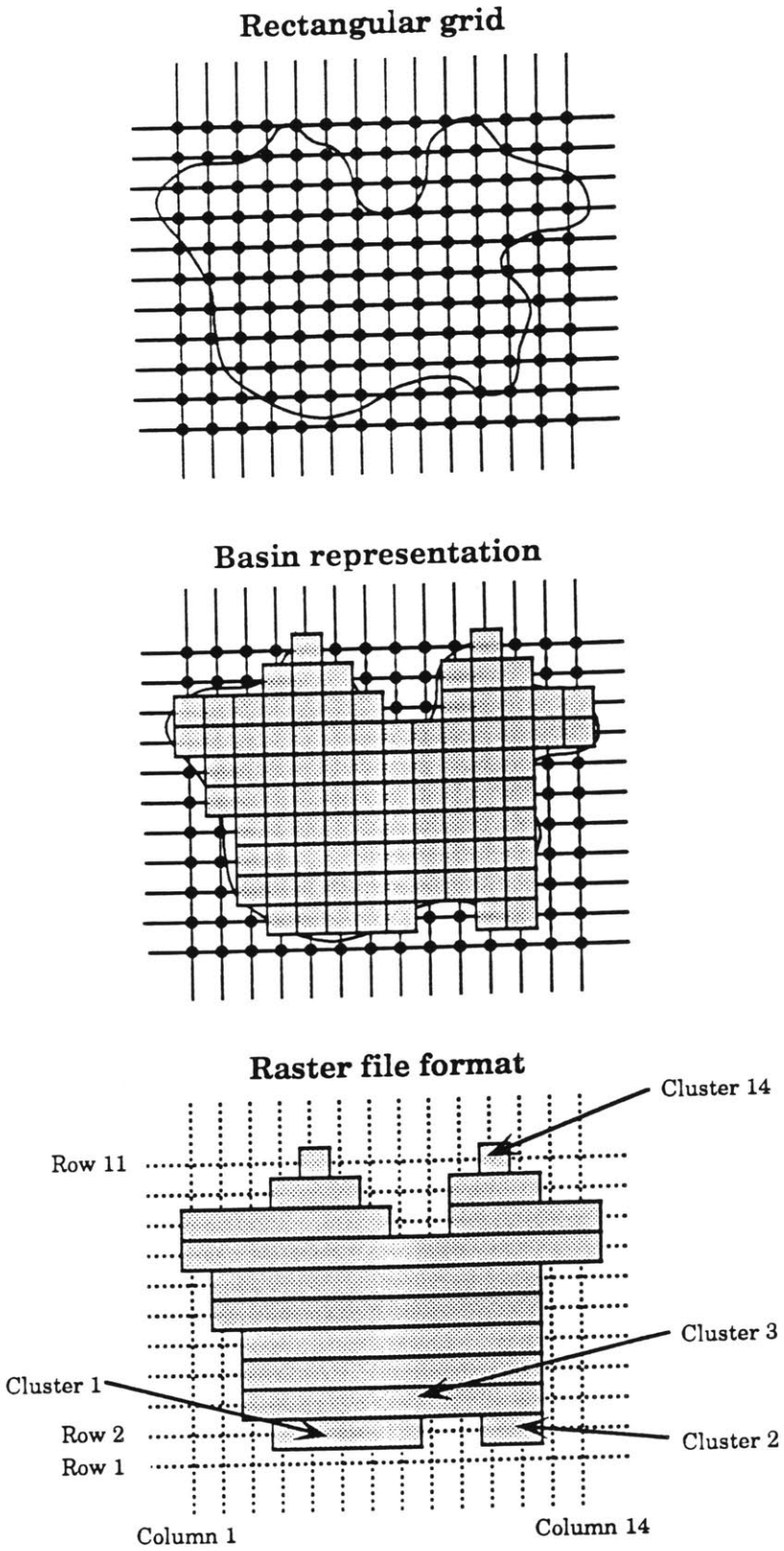


Figure A2.3: *Basin representation in raster format*

A raster file consists of a *header* section, which contains basic information about the localization and spatial structure, and a *data* section, which contains the data. The *header* section is divided in four areas:

- *ASCII header section*: contains general information about the location and coordinate system. Relevant variables are:
  - Cartographic denomination
  - Measurement units of the data
  - Coordinate system
  - Reference of the coordinate system
  - Denomination of agency producing the file
  - East coordinate of the lower-left corner
  - North coordinate of the lower-left corner
  - Date and time of creation of the file
  - Date and time of last update of the file

- *Binary header section*: contains numerical information about technical aspects, such as:
  - Reference value for data section (parameter a)
  - Maximum value of the data section
  - Minimum value of the data section
  - Value to express missing data
  - Number of rows of the map
  - Number of columns of the map
  - X dimension of the cell
  - Y dimension of the cell
  - Data multiplier for integer representation (parameter m)

-*Rows archive section*: contains the list of all the lines that compose the raster. Each register is composed of:

- Identification label of the row
- Pointer to the first cluster in the cluster archive section

- *Cluster archive section*: contains the list of all the cluster that compose the raster. Each register is composed of:

- Identification label of the row
- Number of the first column of the cluster
- Pointer to the first value in the data section
- Number of values in the cluster

The *data* section contains the raw list of all the data about the nodes belonging to the raster, stored in binary integer format. Values are stored consequently, according to the order of the cluster index. Individual values must be accessed through the row and cluster indices.

The following types of files are stored in *Raster* format:

- Geomorphologic description of the basin:
  * Soil types (*sieve.soi* in the example)
  * Pointers (*sieve.ptr*) in the example
  * Slopes (*sieve.slp* in the example)
  * Distance to nearest stream (*sieve.std* in the example)
- Rainfall description: Files in the measured and forecasted rainfall directories. In the example, they are called *sieve.hhhh:mm.mrf* and *sieve.hhhh:mm.frf* respectively

- State variables:
  * Wetting front depth (*sieve.hhhh:mm.snf* in the example)
  * Top front depth (*sieve.hhhh:mm.snt* in the example)
  * Moisture content (*sieve.hhhh:mm.smc* in the example)
  * Runoff generation (*sieve.hhhh:mm.srf* in the example)

Hydrograph format

The Arno project database (Becchi et al., 1991) also contains a standard specification for efficient storage of time series. However, this format was not adopted in RIBS because its design was intended to store long records of a single hydrologic variable at a particular location, not hydrographs corresponding to a storm event in a basin. Furthermore, the data compression strategy was based on the high autocorrelation of long-term hydrologic series, and it was of little use in the RIBS environment, adding a significant computational load. Therefore, a specific format, the hydrograph format, was developed to store RIBS time series in the database.

The *Hydrograph* format is used to store several hydrographs. Each hydrograph is represented as a time series of rainfall and streamflow measurements. The header section contains the number and identification of the hydrographs. The data section contains the values of the time series of rainfall and discharge. Registers contain four fields:

- *Variable identification* (integer from 0 to n-1, with n the number of variables stored in the file)
- *Time tag*, in the format *hhhh:mm.*
- *Value1*, representing streamflow.

- *Value2*, representing rainfall.

Data corresponding to different variables can be stored in any order, as long as they keep temporal precedence for every variable. An example of the file format, containing three hydrographs generated for the Fornacina gauge during the storm of November 1991 in the Sieve basin, can be seen in Figure A2.4.

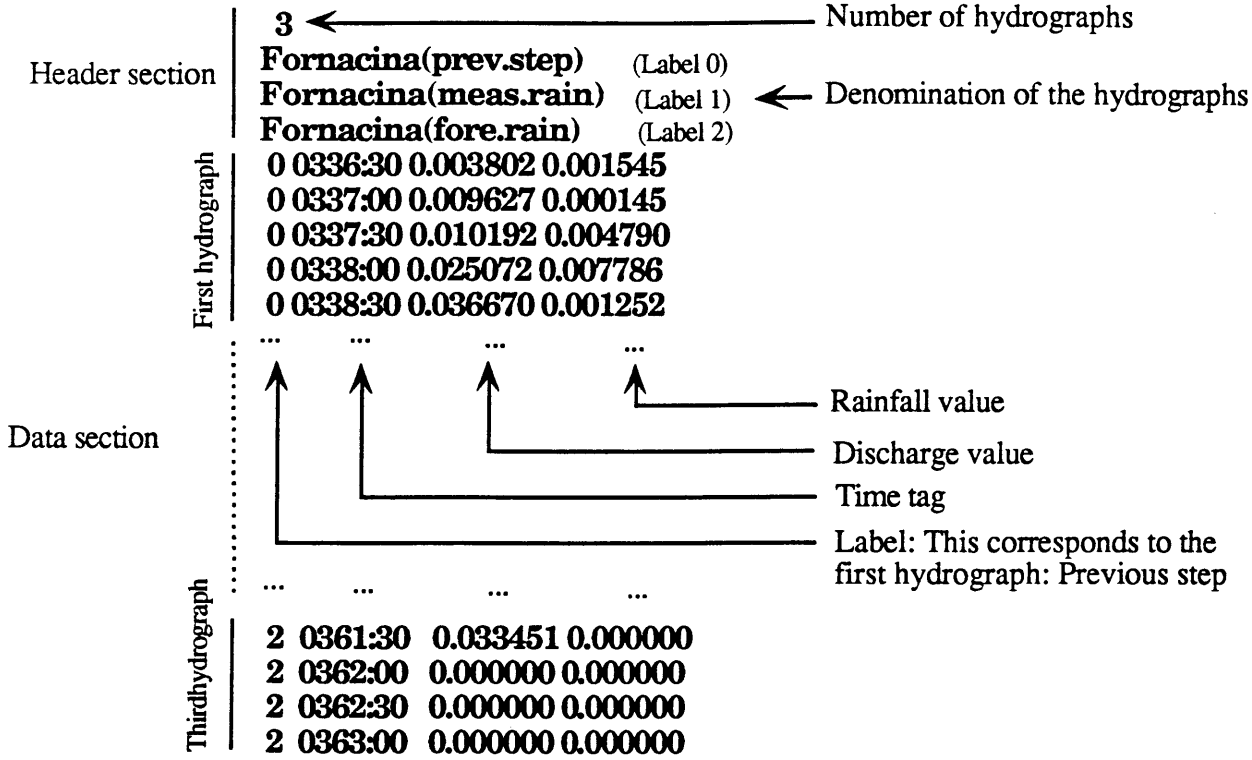


***Figure A2.4** Example of the Hydrograph file format*

Pixel format

The *Pixel* file format contains the data required to define pixel state at a given time: static properties and state variables. It is composed of an identification line, containing the denomination of the pixel, and a list of

pixel variables, in this order:

- Water table depth N_{wt} (mm)
- Current wetting front depth N_f (mm)
- Current top front depth N_t (mm)
- Current moisture content of the unsaturated zone M_u (mm)
- Current total moisture content M_t (mm)
- Slope $tan(\alpha)$
- Parameter ε
- Parameter f (mm^{-1})
- Saturation moisture content θ_s
- Residual moisture content θ_r
- Anisotropy ratio a_r
- Normal hydraulic conductivity at the surface K_{0n} (mm/h)

Figure A2.5 contains an example of the Pixel format, generated by RIBS for the Sieve case study.

i41j87.1014:00.pxl ←	Denomination
1113.000000 ←	Water table depth Nwt (mm)
1113.000000 ←	Wetting front depth Nf (mm)
0.000000 ←	Top front depth Nt (mm)
7.616391 ←	Unsaturated moisture content Mu (mm)
11.546391 ←	Total moisture content Mt (mm)
0.327000 ←	Slope (tan α)
3.500000 ←	Parameter ε
0.000700 ←	Parameter f (mm-1)
0.520000 ←	Saturation moisture content θs
0.064000 ←	Residual moisture content θr
10.000000 ←	Anisotropy ratio ar
16.600000 ←	Surface hydraulic conductivity Kon (mm h-1)

***Figure A2.5** Eample of the Pixel file format*

A2.2.2 Trace formats

Trace files contain secondary output from the model. They are intended for detailed analysis of model behavior, mainly for debugging purposes. Trace files are not connected to any type of graphical presentation within RIBS, and their design goal is to offer large amounts of information, rather than to obtain easy readability. They are therefore hard to interpret by the non-specialized user. Three types of trace files are generated by RIBS: pixel traces, basin traces and gauge traces, which are revised as follows.

Pixel trace

The pixel trace contains information about pixel variables for every computation time step. It is intended for debugging purposes (detailed analysis of pixel evolution when unexpected behavior is detected) and therefore the variables included in the output can and should change frequently during system development. The current version includes the following variables:

- Basin array index of the pixel
- Wetting front depth (mm)
- Wetting front speed in the time step (mm/h)
- Top front depth (mm)
- Top front speed in the time step (mm/h)
- Moisture content (mm)
- Moisture content in the unsaturated zone (mm)
- Subsurface inflow rate (mm/h)

- Subsurface outflow (mm/h)
- Rainfall rate (mm/h)
- Equivalent rainfall rate (mm/h)
- Saturation level corresponding to the equivalent rainfall rate (mm)

A line containing the above variables is printed every computation time step for pixels in the trace list.

Basin trace

Basin traces provide information about how runoff is generated in the basin. Two files are created. The first file contains information about number of pixels in each state. The second file contains information about runoff volume generated by pixels in each state. In both files, one line containing a list of variables is written at the end of every time step.

For the *number* file the list of variables is:

- Number of pixels with water table at the surface
- Number of pixels with water table at the surface generating return flow
- Number of stream pixels
- Number of stream pixels generating return flow
- Number of unsaturated pixels
- Number of unsaturated pixels generating infiltration-excess runoff
- Number of perched-saturated pixels
- Number of perched-saturated pixels generating infiltration-excess runoff

- Number of surface-saturated pixels
- Number of surface-saturated pixels generating infiltration runoff
- Number of surface-saturated pixels generating return flow
- Number of fully-saturated pixels
- Number of fully-saturated pixels generating return flow

For the *volume* file the list of variables is:

- Volume of infiltration-excess runoff generated by pixels with water table at the surface in mm
- Volume of return flow generated by pixels with water table at the surface in mm
- Volume of infiltration-excess runoff generated by stream pixels in m m
- Volume of return flow generated by stream pixels in mm
- Volume of infiltration-excess runoff generated by unsaturated pixels in mm
- Volume of infiltration-excess runoff generated by perched-saturated pixels in mm
- Volume of infiltration-excess runoff generated by surface-saturated pixels in mm
- Volume of return flow generated by surface-saturated pixels in mm
- Volume of infiltration-excess runoff generated by fully-saturated pixels in mm
- Volume of return flow generated by fully-saturated pixels in mm

Gauge trace

The gauge trace contains information about the decomposition of the total hydrograph in the different types of runoff generation modes. The register of the gauge trace contains five fields. The first field is the time reference, in the format *hhhh:mm*. The other four fields correspond to the four hydrographs generated:

- Return flow in permanently saturated pixels in m^3/s
- Return flow in temporary saturated pixels in m^3/s
- Infiltration-excess runoff in permanently saturated pixels in m^3/s
- Infiltration-excess runoff in temporary saturated pixels in m^3/s

A2.2.3 Auxiliary files

Auxiliary files contain additional information used by the RIBS package. The program that generates rainfall distribution from raingauge measurements uses to input files, and two other files are needed by the rainfall-runoff module.

Raingauge description

The raingauge description file contains the list of raingauges used to generate the spatial distribution of rainfall. Each line of the file contains the following variables:

- Raingauge identification label: integer number.
- *X* coordinate of the raingauge location (in m)
- *Y* coordinate of the raingauge location (in m)

- Raingauge denomination

Figure A2.6 shows an example of the raingauge identification file, *raingauges.tab*.

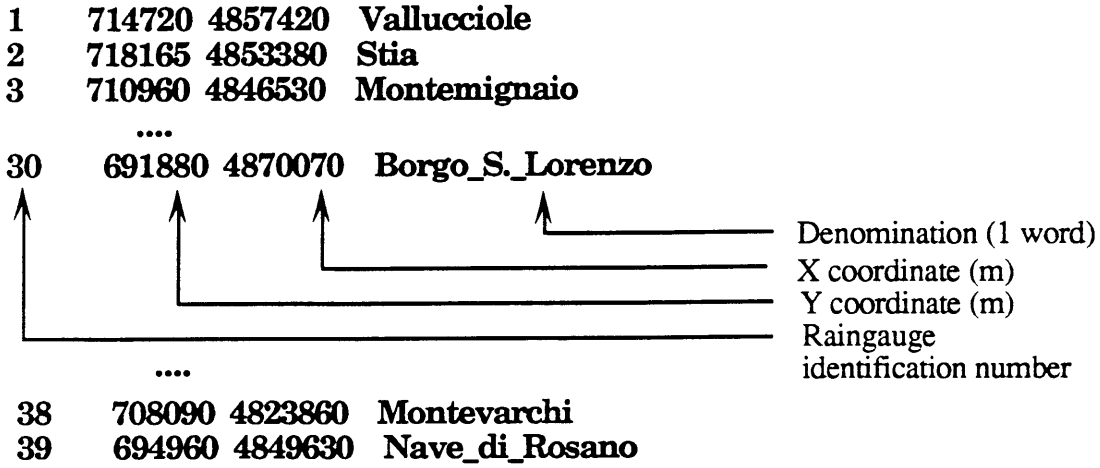


***Figure A2.6** Example of the raingauge identification file*

Rainfall data

The rainfall data file contains the readings of the raingauges. It is used by the rainfall acquisition module to emulate the arrival of distributed rainfall information. For each time step, the rainfall data contains three sections: the time identification section, the raingauge identification section and the rainfall values section. The time identification section contains the following fields:

- The keyword '*RG*' to identify raingauge rainfall data (in case the application is expanded to accept other types of input)
- Time tag, in the format *hhhh:mm*, which corresponds to the time at which the measurement is generated.

- Number of raingauges with valid reading in this time step.
- Duration of the rainfall in this time step (min)

The raingauge identification section contains a list of the identification labels of the raingauges which obtained valid readings in this time step. The rainfall values section contains the readings of the raingauges in the same order as they appear in the previous section. The reading is expressed as rainfall depth in mm. Figure A2.7 shows an example of rainfall data file, corresponding to the storm of November 1991 for the Sieve basin, *rain91.chn*.

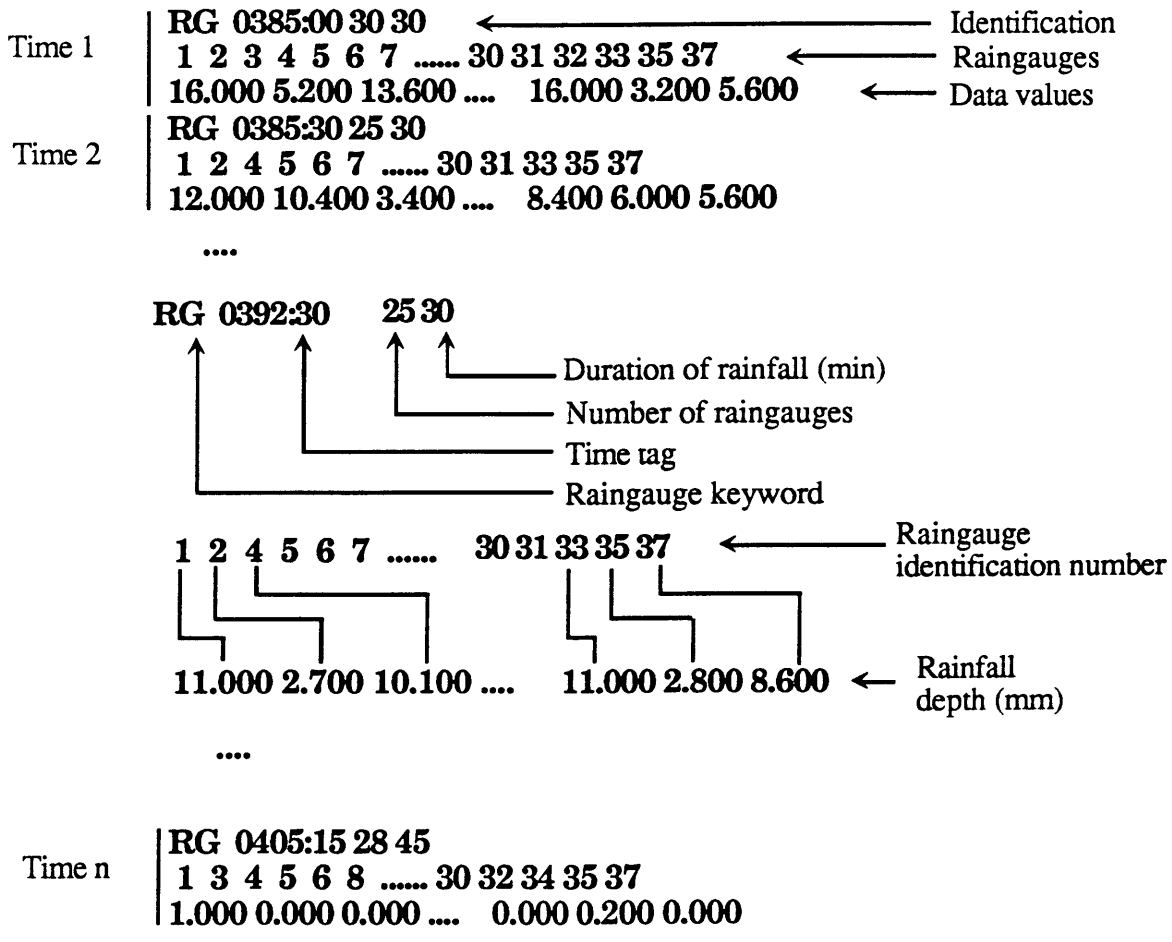


***Figure A2.7** Example of the rainfall data file*

Soil types

The soil types file contains the numerical values of the Brooks-Corey parameters for the soil classes. The first line of the file contains the number of soil classes, and then each register contains a soil class, with the variables:

- Normal hydraulic conductivity at the surface K_{0n} (mm/h)
- Saturation moisture content θ_s
- Residual moisture content θ_r
- Parameter ε

Figure A2.8 shows an example of the soil types file used in the Sieve case study, *sieve.sdat*.

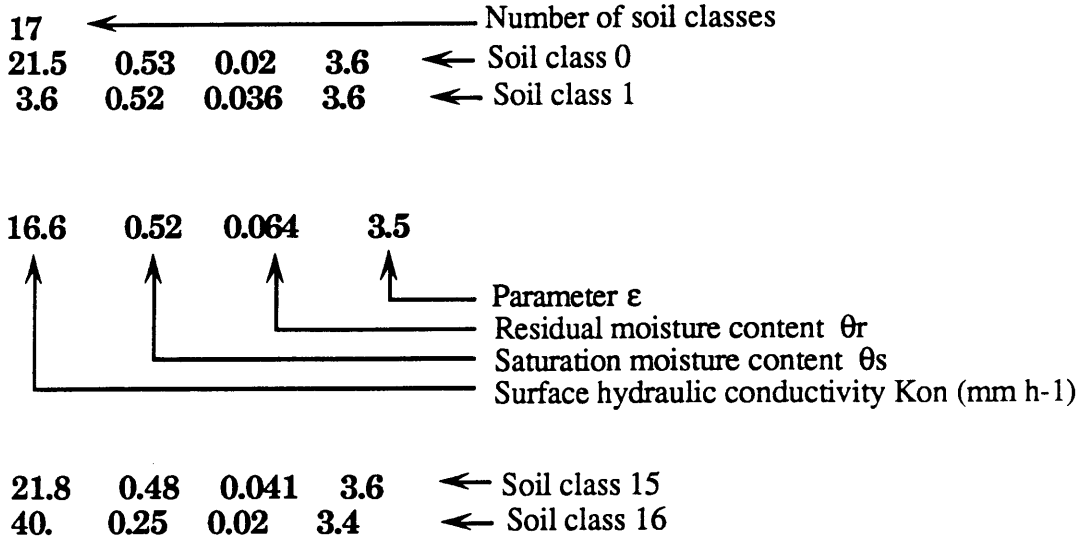


***Figure A2.8** Example of the soil types file format*

Order of computations

The order of computations file contains a list of the basin pixels on the order in which they should be evaluated in the basin loop. The first two fields in the file correspond to number of hillslope pixels and number of stream pixels. Then, each pixel is presented in a register containing the row index and the column index. The order of computations file may be generated using a program which reads the *pointer* raster file and generates the list of pixels applying the recursive relation 'drains to'. Figure A2.9 shows the order of computation file corresponding to the Sieve basin: *sieve.ord*.

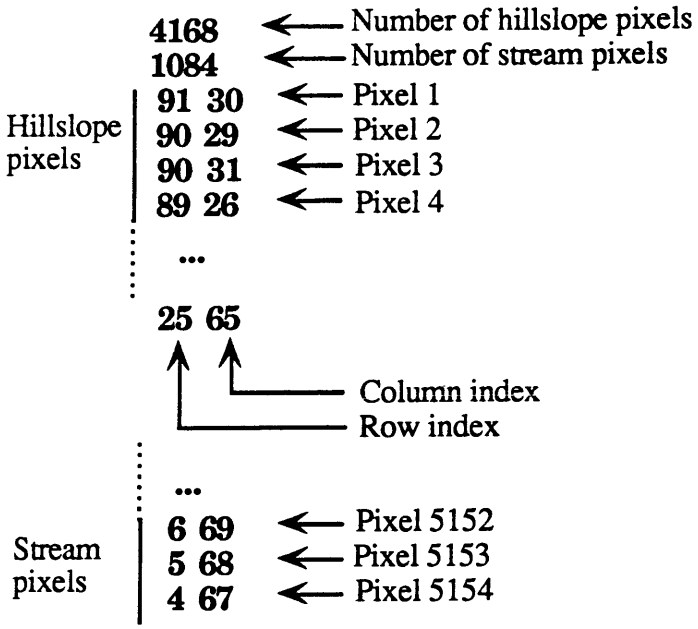


***Figure A2.9** Example of the order of computations file format*

APPENDIX 3

RIBS User Manual

This appendix constitutes the user manual for the RIBS package. The core of RIBS is composed of three executable modules: a rainfall acquisition module (*rain_gen_gaug*), a rainfall-runoff module (*dbsim*) and a basin viewer (*x_bviewer*). The executable modules use three other graphic programs: the raster viewer (*x_rasgraf*), the hydrograph viewer (*x_hydgraf*) and the pixel viewer (*x_pixgraf*), which can also be executed independently by the user. There are also several auxiliary programs to facilitate handling of distributed data (*gtr*, *mtr* and *rtm*).

Executable modules in RIBS read information from the *unix* environment, from arguments in the command line and from data files. Before starting a module, the user must define the adequate data files and set the required environmental variables. The definition of environmental variables can be done directly on the computer terminal, but it is usually more convenient to prepare a script file with the definition of the environmental variables and the system call to the executable module. Therefore, a typical run of a RIBS module consists simply of running the shell script previously prepared for it. This appendix contains information about the contents of the script files necessary to run the modules in their different modes of operation. For every module we provide information about the necessary input files, the program call with their arguments and the interactive behavior.

A number of examples are presented to illustrate the practical use of the model. They are built using the database available for the Sieve basin, already introduced in Appendix 2 (Figure A2.2). All files in the example are under the directory "*/model*", which represents the home directory of the model user. Executable modules are stored in "*/model/bin*", and data files are stored in "*/model/data*". Geomorphologic data correspond to the Sieve basin, and are stored in "*/model/data/sieve*". This directory also includes a subdirectory, *iniwt*, where several files containing initial water table levels in the basin are stored. All examples correspond to the storm of November, 1991, and their results are stored in the directory "*/model/data/st91*", under five different subdirectories: *mrain*, *frain*, *stvar*, *hydrg* and *rainaux*.

The starting point to run RIBS applications is the following list of input and data files:

- Rainfall input files, stored in *st91/rainaux*:
    * rainfall data file: *rain91.chn*, ASCII file.
    * raingauge identification list: *raingauges.tab*, ASCII file.
- Geomorphologic input files, stored in *sieve*:
    * soil types: *sieve.soi*, raster file.
    * pointers: *sieve.ptr*, raster file.
    * distance to stream: *sieve.std*, raster file.
    * slopes: *sieve.slp*, raster file.
    * soil data: *sieve.sdat*, ASCII file.
    * order of computations: *sieve.ord*, ASCII file.

- Initial state files, stored in *sieve/iniwt*:

  * water table depth: *sieve.47jan50ar.nwt*, raster file.

Other files included in Figure A2.2 correspond to results generated by the RIBS package. Input files in ASCII format can be created with any standard text editor, using the format descriptions provided in Appendix 2. The generation of input files in raster format requires specific software. The RIBS package provides three auxiliary programs to create raster files from matrix descriptions and to transfer data from matrix to raster format and back. These programs are described first. Then, the usage of the rainfall acquisition and the rainfall-runoff modules is discussed. Finally, the interactive programs of the user interface are described.

A3.1 Raster file handling

The three following programs are available to manipulate data in raster format:

- Program *gtr* (grid to raster), to create the raster format file corresponding to a basin discretization on a rectangular grid.
- Program *mtr* (matrix to raster), to create a raster file with the data contained in an ASCII matrix file.
- Program rtm (raster to matrix), to create an ASCII matrix file with the data contained in a raster file.

To create raster files for a given basin the user follows a two-step procedure. The user must run the *gtr* program to create the first file in raster format for the basin, and then use matrix descriptions of the

distributed variables and the raster template file to generate raster files of the variables with the program *mtr*.

A3.1.1 The program *gtr*

The program *gtr* generates a raster file description of a basin. The objective is to create a *Raster header* structure, with all the generic information corresponding to basin shape, location and coordinate system. *gtr* defines the rows and clusters corresponding to the basin boundaries, but it leaves the values section empty, since only file structure is of concern.

The operation of the *gtr* program is represented in Figure A3.1. The user must generate an input file describing the basin shape in a matrix whose rows and columns follow the rectangular grid. The file must contain the number of rows and columns of the matrix, pixel size and coordinates of the lower left pixel. Basin shape is described by assigning different values to matrix cells located inside and outside the basin boundaries. Cell values are defined by the user.

The standard format to call *gtr* is:

% gtr *BasinMask RasterFileName vmin vmax option*

- *BasinMask* is the name of the file describing the basin shape.
- *RasterFileName* is the name of the raster file to be generated.
- *vmin* and *vmax* are the minimum and maximum values of cells inside the basin boundaries. *gtr* considers that all matrix elements with values between *vmin* and *vmax* belong to the basin. Matrix elements

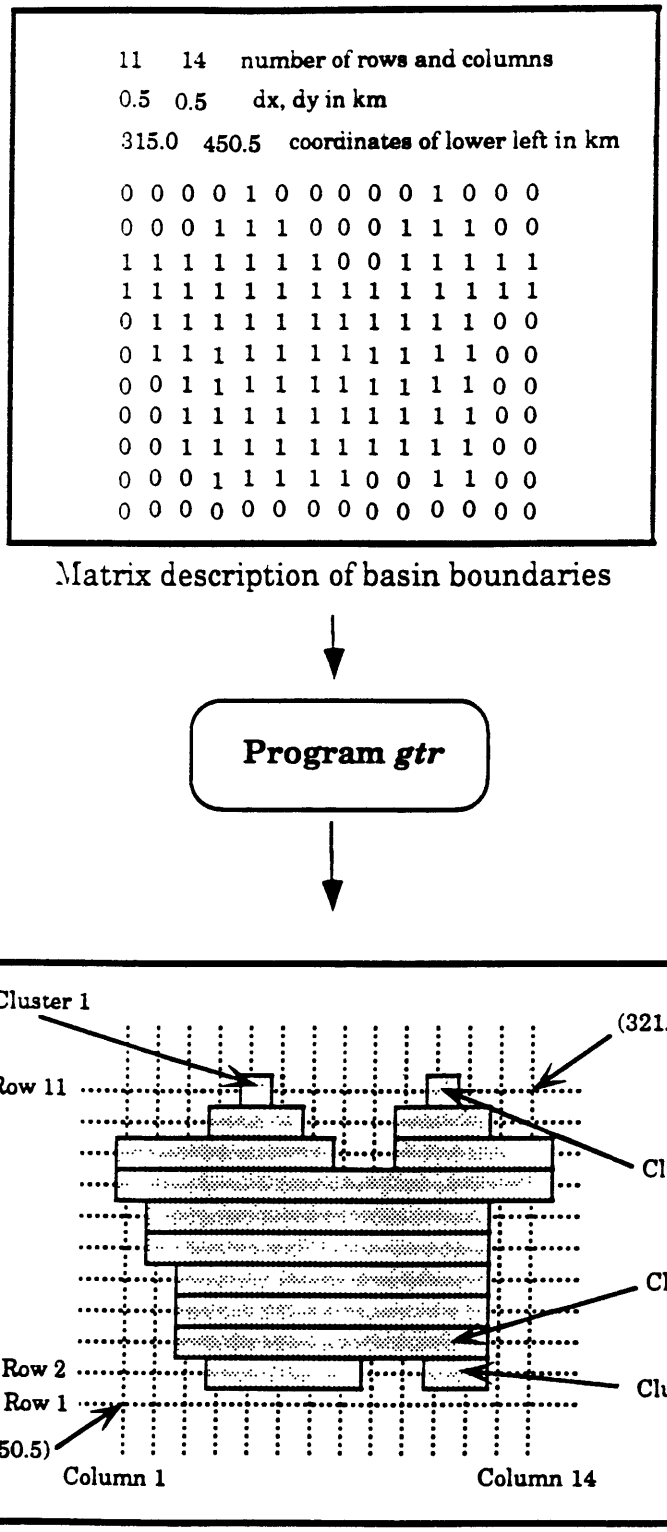


Figure A3.1: *Schematic operation of the program gtr(grid to raster), used to transform grid basin data into a template raster file*

with values smaller than *vmin* or greater than *vmax* are considered to be outside the basin boundaries.

- *option* describes the format of the input file, according to the following keywords:

  * *a* : input file is in ASCII format

  * *f* : input file is in floating point format

  * *s* : input file is in short integer format

  * *l* : input file is in long integer format

For instance, if *sieve.mask* is an ASCII file (similar to that in the upper part of Figure A3.1) containing the description of the Sieve basin in terms of zeros (outside) and ones (inside), the call:

```
% gtr sieve.mask sieve.ras 1 1 a
```

would generate the raster file *sieve.ras*, with the structure of the Sieve basin.

A3.1.2 The program mtr

The program *mtr* generates a data file in raster format given an ASCII file containing the data and a raster file template corresponding to the same basin. If no raster file is previously available for the basin, it can be generated using the program *gtr*.

The operation of the *mtr* program is represented in Figure A3.2. The user must provide a raster template of the basin and a file with the data in matrix format. The data file must specify the number of rows and column of the matrix and the cell size. Matrix cells outside the boundaries of the basin are ignored. The user can reserve a special value to represent

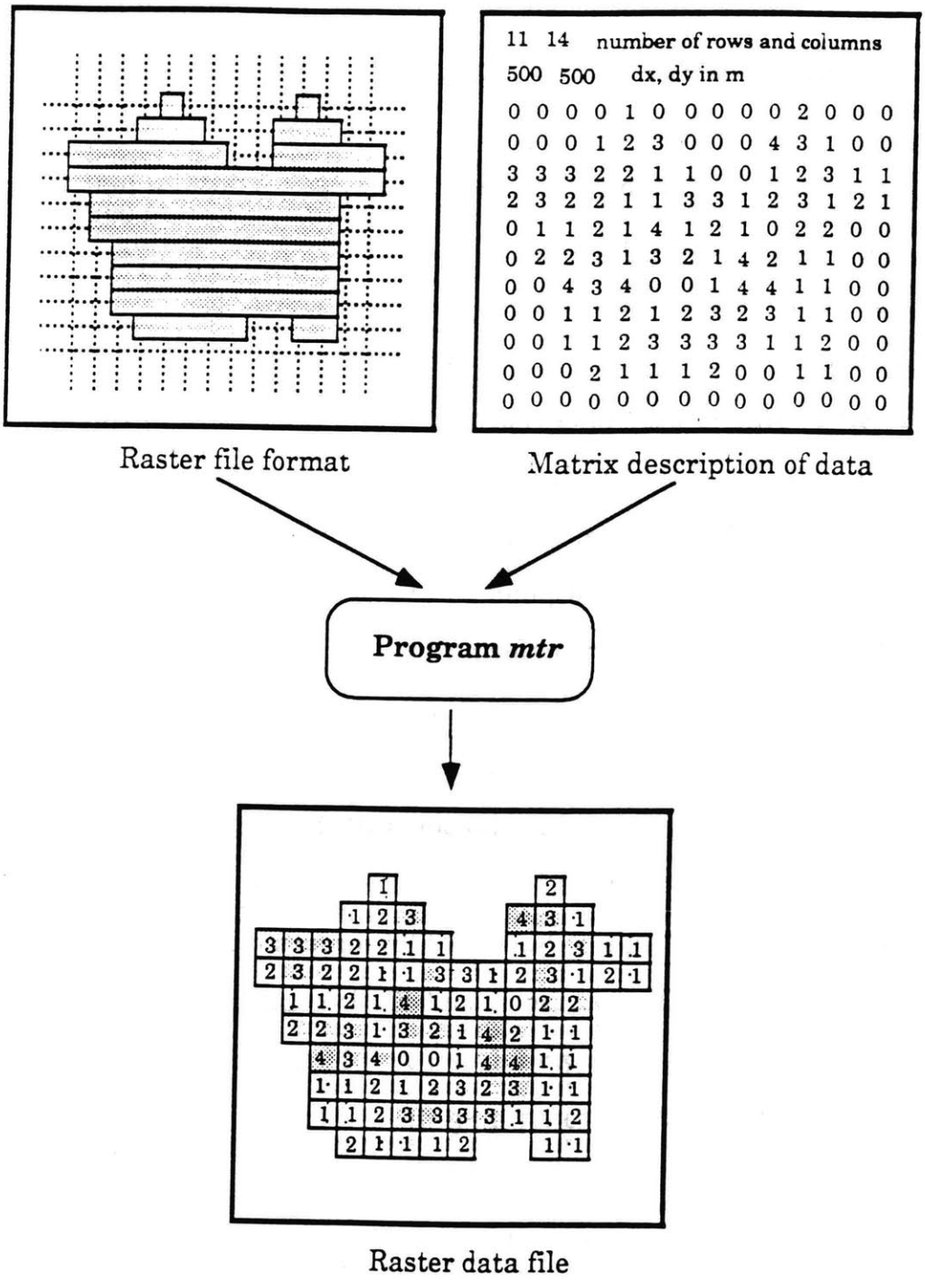


Figure A3.2: *Schematic operation of the program mtr: matrix to raster, used to transform a matrix file into a raster file*

missing data inside the basin boundaries. All other values are stored in the output raster file.

The standard format to call *mtr* is:

% mtr *MatrixFile RasterMask RasterOutput MissingValue* [*mult*] [*shift*]

- *MatrixFile* is the name of the file containing the data in matrix format.
- *RasterMask* is the name of a raster file corresponding to the same basin. It is used as a template to store the new data.
- *RasterOutput* is the name of the raster file to be generated.
- *MissingValue* is the value representing missing data in the matrix.
- *mult* is the exponent m in Equations A2.1a and A2.1b. If *mult* is not specified, it is taken as 0, and data are stored with their original values. *mult* must correspond to the order of magnitude of the data values. If, after the transformation, an overflow of the maximum storage value of 32767 occurs, the program prints an error message and stops. No output file is generated.
- *shift* is the coefficient a in Equations A2.1a and A2.1b. If *shift* is not specified, it is taken as 0. If, after the transformation, an overflow of the maximum storage value of 32767 occurs, the program prints an error message and stops. No output file is generated.

For instance, if *soils.grid* is an ASCII file (similar to that in the upper part of Figure A3.2) containing the soil types for the Sieve basin, the call:

% mtr soils.grid sieve.ras sieve.soi -1

would generate the raster file *sieve.soi*, with the soil types of the Sieve basin. Missing values are represented by -1, and no transformation is applied to the data, since neither *mult* nor *shift* are defined.

A3.1.3 The program rtm

The program *rtm* generates a data file in matrix format corresponding to a given raster file. The purpose of *rtm* is to offer editing possibilities for raster files. If the user wishes to change some values in a raster file, he or she may create a matrix version with *mtr*, edit the resulting ASCII file to make the changes, and generate the raster file again with *mtr*.

The operation of the *rtm* program is represented in Figure A3.3. Given a raster file, *rtm* generates a file with a matrix description of the data. The resulting file can be used directly as input to *mtr*. Matrix cells outside the boundaries of the basin are assigned the missing data value.

The standard format to call *rtm* is:

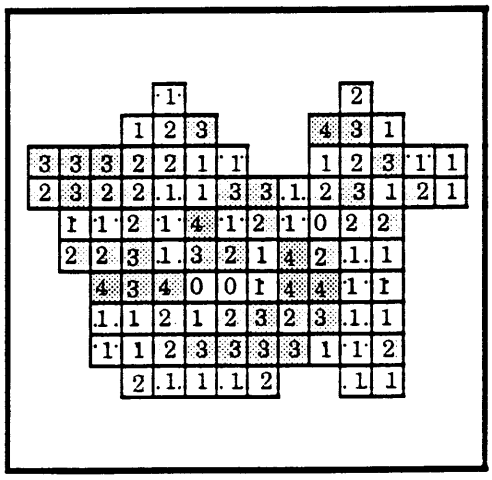
```
% mtr RasterFile MatrixOutput MissingValue
```

- *RasterFile* is the name of the input file, in raster format.
- *MatrixOutput* is the name of the output file containing the data in matrix format.
- *MissingValue* is the value assigned to the empty cells.

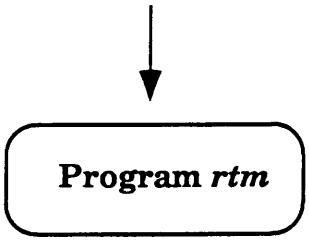
For instance, given the *sieve.soi* raster file, an ASCII file (similar to that in the lower part of Figure A3.3) containing the soil types for the Sieve basin may be generated with the call:

```
% rtm sieve.soi soils.grid.new -3
```

The file *soils.grid.new* contains a matrix description of the soil types of the Sieve basin. Missing values are represented by -3.



Raster data file



11 14 number of rows and columns
500 500 dx, dy in m

-3	-3	-3	-3	1	-3	-3	-3	-3	-3	2	-3	-3	-3
-3	-3	-3	1	2	3	-3	-3	-3	4	3	1	-3	-3
3	3	3	2	2	1	1	-3	-3	1	2	3	1	1
2	3	2	2	1	1	3	3	1	2	3	1	2	1
-3	1	1	2	1	4	1	2	1	0	2	2	-3	-3
-3	2	2	3	1	3	2	1	4	2	1	1	-3	-3
-3	-3	4	3	4	0	0	1	4	4	1	1	-3	-3
-3	-3	1	1	2	1	2	3	2	3	1	1	-3	-3
-3	-3	1	1	2	3	3	3	3	1	1	2	-3	-3
-3	-3	-3	2	1	1	1	2	-3	-3	1	1	-3	-3
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3

Matrix description of data

Figure A3.3: *Schematic operation of the program rtm: raster to matrix, used to transform a raster file into a matrix file*

A3.2 Rainfall acquisition module

In this prototype implementation of RIBS, the rainfall acquisition module is substituted by the program *rain_gen_gauge*. *rain_gen_gauge* simulates the arrival of rainfall in time taking raingauge measurements from a file and generating raster files with the spatial distribution of rainfall in the basin. It also generates rainfall forecasts based on a simple *AR(1)* model. *rain_gen_gauge* can optionally display measured and forecasted rainfall and hyetographs of rainfall registered in several raingauges. The rainfall acquisition module needs two data files: the rainfall data file and the raingauge identification. Format for both files is described in Appendix 2.

The standard format to call *rain_gen_gaug* is:

```
% rain_gen_gaug RasterMask RainChannel Raingauges exponent [GaugeList]
```

- *RasterMask* is any raster file corresponding to the same basin boundaries. It is used by *rain_gen_gaug* as a template to generate rainfall in the same raster format (boundaries and spatial discretization).
- *RainChannel* is the name of the file which contains the rainfall data, as described in Appendix 2.
- *Raingauges* is the name of the file which contains the description of raingauges: identification label and denomination, as described in Appendix 2.
- *exponent* is the absolute value of the negative exponent in the power law governing the interpolation of rainfall. If *exponent* is given a very

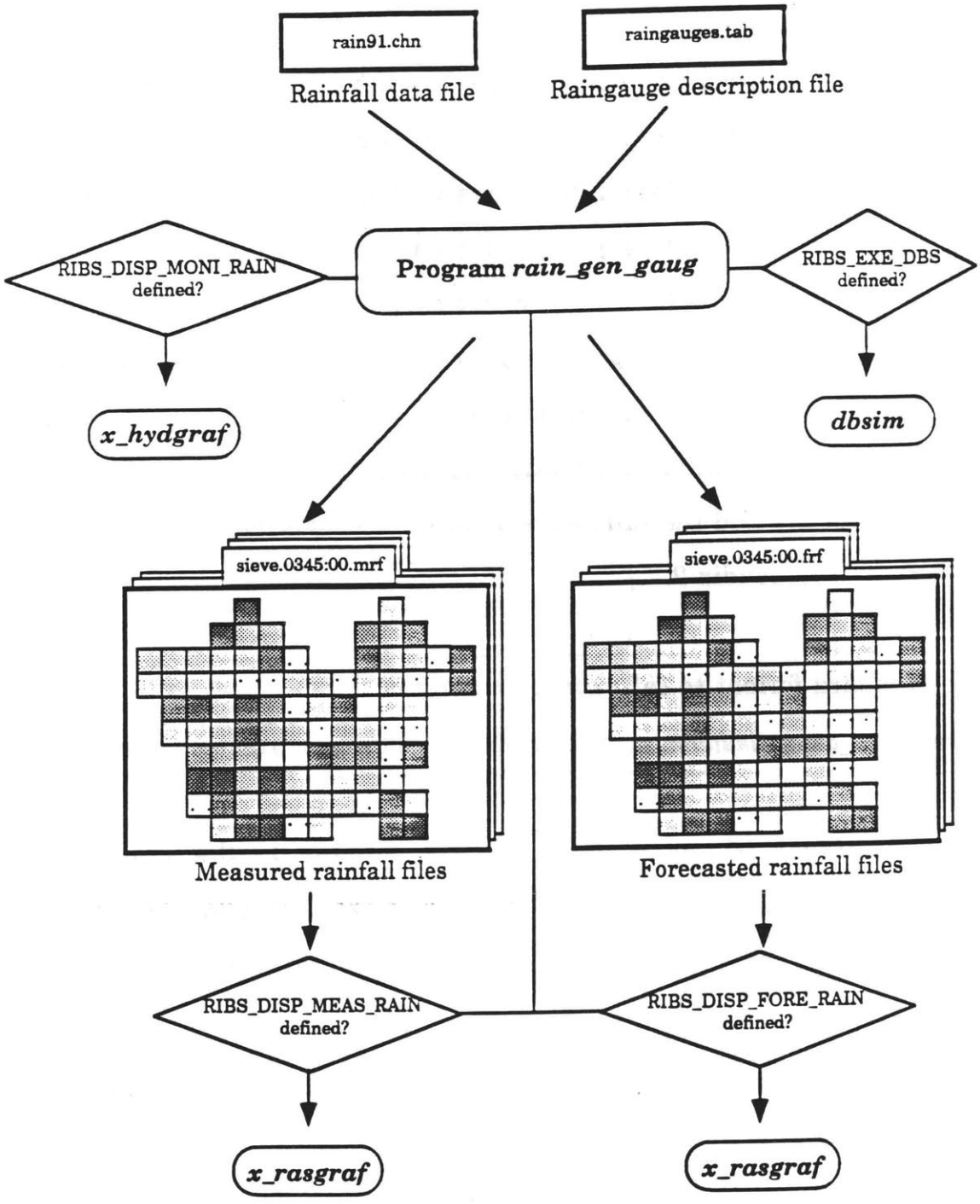


Figure A3.4: *Schematic operation of the program rain_gen_gauge, used simulate the arrival of distributed rainfall information*

large value, a distribution similar to the Thiessen polygons is obtained. If *exponent* is zero, rainfall is uniformly distributed in the basin, and equal to the average of the raingauges involved.

- *GaugeList* is the list of raingauges to appear in the hyetograph report, if activated. The format for the list if *key1 key2 ... keyn*, where *keyi* is the identification label of raingauge *i*.

rain_gen_gauge also takes information from environmental variables, which should be defined in the script file, before calling the main program. Environmental variables define paths and file name descriptions for output files and offer information about which processes should be activated. The following variables control the conditional activation of child processes.

- *RIBS_EXE_DBS* If this variable is defined, the rainfall-runoff module *dbsim* is activated by *rain_gen_gauge*. In this case, *rain_gen_gauge* notifies *dbsim* every time it generates new rainfall information.If this variable is not defined, RIBS is run only in rainfall-generation mode. Rainfall files are generated, but no rainfall-runoff modeling is performed.
- *RIBS_DISP_FORE_RAIN* If this variable is defined, the on-line presentation of forecasted rainfall is activated by *rain_gen_gauge*. In this case, a *x_rasgraf* process is started by *rain_gen_gauge*. The process creates a raster viewer window which presents the updated forecasted rainfall every time *rain_gen_gauge* generates new rainfall information.
- *RIBS_DISP_MEAS_RAIN* If this variable is defined, the on-line presentation of measured rainfall is activated by *rain_gen_gauge*. In

this case, a *x_rasgraf* process is started by *rain_gen_gauge*. The process creates a raster viewer window which presents the updated measured rainfall every time *rain_gen_gauge* generates new rainfall information.

- *RIBS_DISP_MONI_RAIN* If this variable is defined, the on-line presentation of hyetographs is activated by *rain_gen_gauge*. In this case, a *x_hydgraf* process is started by *rain_gen_gauge*. The process creates a raster viewer window which presents the updated hyetographs for the selected raingauges every time *rain_gen_gauge* generates new rainfall information.

The operation of rain_gen_gauge is summarized in Figure A3.4. By activating or inhibiting the child processes, the user can operate the rainfall acquisition module in different modes: with or without graphic display of results and with or without connection to the rainfall-runoff transformation module.

The following shell script can be used to activate the rain_gen_gauge in rainfall-generation mode in the example. Note that path, root and extension of the file names are defined through environmental variables. Environmental variables are also used to activate presentation processes.

```
setenv  RIBS_T_GENE_START  0336:00
setenv  RIBS_T_BEGIN  0336:00
setenv  RIBS_T_GENE_END    0424:00

setenv  RIBS_PATH_MEAS_RAIN  /model/data/st91/mrain
setenv  RIBS_PATH_FORE_RAIN  /model/data/st91/frain
setenv  RIBS_PATH_HIST_RAIN  /model/data/st91/rainaux

setenv  RIBS_ROOT_MEAS_RAIN  sieve
setenv  RIBS_ROOT_FORE_RAIN  sieve
setenv  RIBS_ROOT_HIST_RAIN  rain

setenv  RIBS_EXT_MEAS_RAIN  mrf
```

```
setenv  RIBS_EXT_FORE_RAIN  frf
setenv  RIBS_EXT_HIST_RAIN  hst

setenv  RIBS_DISP_MEAS_RAIN  "/model/bin/x_rasgraf  -N25  -
C/model/data/fabio.pal      \
-X  0.0001  10.0  -Q  -n  Measured_Rain"

 setenv  RIBS_DISP_FORE_RAIN  "/model/bin/x_rasgraf  -N25  -
C/model/data/fabio.pal   \
-X  0.0001  10.0  -Q  -n  Forecasted_Rain"

 setenv  RIBS_DISP_MONI_RAIN  "/model/bin/x_hydgraf  -Q  -n  Raingauges"

/model/bin/rain_gen_gaug  /model/data/sieve/geomr/sieve.soi        \
/model/data/st91/rainaux/rain91.chn        /
/model/data/st91/rainaux/raingauges.tab  2.0  22  30
```

Execution of this script file generates a list of raster files in the directories */model/data/st91/mrain* (measured) and */model/data/st91/frain* (forecasted), containing the rainfall intensity distribution which corresponds to the rainfall data stored in *rain91.chn*. Rainfall intensity is generated from raingauge data using a weighting scheme based on the inverse of the distance squared. The program also generates a list of files in the directory */model/data/st91/rainaux* containing hyetographs of the rainfall registered in gauges number 22 and 30. Files names are generated with the format: *sieve.hhhh:mm.mrf*, with the time tag *hhhh:mm* taken from the input data file *rain91.chn*.

Since the variable *RIBS_EXE_DBS* is not defined, no rainfall-runoff modeling is performed. The variables *RIBS_DISP_MEAS_RAIN*, *RIBS_DISP_FORE_RAIN* and *RIBS_DISP_MONI_RAIN* are defined, and therefore the corresponding graphic presentation processes are activated. Forecasted and measured rainfall distributions appear in two independent windows containing raster viewers. Another window contains a hydrograph viewer which presents hyetographs.

A3.3 The rainfall-runoff transformation module

The rainfall runoff transformation module, *dbsim*, takes rainfall information and generates model results. It can be operated in several different modes, controlled by the arguments in the command line and by the setting of environmental variables. Input files are located in three directories: the geomorphology directory, which stores permanent basin information, the measured rainfall directory and the forecasted rainfall directory.

The following input files are needed by the rainfall-runoff module:

- Geomorphologic files: soil types, pointers, distance to stream, distance to gauge, slopes, soil data, order of computations and initial water table.
- Rainfall files: measured and forecasted rainfall.

The rainfall-runoff module can generate raster files with the evolution of basin state variables and hydrograph files with streamflow forecasts at several points in the basin. In certain modes of operation, the generation of intermediate results is optional.

The standard format to call *dbsim* is:

% dbsim [-A] [-R] [-F] [-S*hhhh:mm*] [-G*n gauge1 ... gaugen*]

The arguments in brackets are optional. Arguments in cursive stand for text which must be provided by the user. All other arguments must be typed textually in order to activate the corresponding mode of operation. The interpretation of the arguments is the following:

-F : Forecasting mode. The -F option activates forecasting mode. The model reads both measured and forecasted rainfall and performs the

full forecasting loop. It can be used either on-line (connected with the rainfall-acquisition module) or off-line.

-*A : Automatic rainfall input*. Rainfall information is taken from the files existing in the measured and forecasted rainfall directories, instead of from the standard input. This option is used for off-line simulations, when the whole storm is previously available and *dbsim* is not used in connection with the rainfall acquisition module.

-*R : Intermediate result writing*. When the model is run in simulation mode (no forecasted rainfall computation) and graphic presentations are not activated, intermediate basin state and result files are not required for model operation, and therefore they are not generated. In these conditions, the model writes intermediate results only if the -R option is activated.

-*S : Initial state*. The -S option allows model initialization with an intermediate basin state. It must be followed by the time tag of the state, in the format *hhhh:mm*. Basin state files and hydrographs should be available for that time tag. Hydrographs should correspond exactly to the same model parameters (gauges, result time step and initial time) as the present run. This option can be used off-line or to resume an interrupted computation on-line.

-*G : Gauge declaration*. This option lets the user define the points in the basin for which hydrographs are to be generated. The format is -G*n rowg0 columng0 .. columngn*, where *n* is the number of gauges required, and *rowgi* and *columngi* are the row and column numbers of gauge *i*. Gauge denominations must be defined in the corresponding environmental variables. If the -*G* argument is not specified, the model assumes one single gauge located at the final basin outlet.

In addition to the command line, *dbsim* also takes information from the environmental variables defined in the script. It reads paths and file name descriptions for input and output files, obtains model and simulation parameters and gets information about which processes should be activated, checking the following variables.

- *RIBS_DISP_NF* If this variable is defined, the on-line presentation of wetting front depth is activated by *dbsim*. In this case, a *x_rasgraf* process is started by *dbsim*. The process presents the updated wetting front depth every time *dbsim* writes intermediate results.
- *RIBS_DISP_NT* If this variable is defined, the on-line presentation of top front depth is activated by *dbsim*. In this case, a *x_rasgraf* process is started by *dbsim*. The process presents the updated top front depth every time *dbsim* writes intermediate results.
- *RIBS_DISP_MC* If this variable is defined, the on-line presentation of moisture content is activated by *dbsim*. In this case, a *x_rasgraf* process is started by *dbsim*. The process presents the updated moisture content every time *dbsim* writes intermediate results.
- *RIBS_DISP_RF* If this variable is defined, the on-line presentation of runoff generation rate is activated by *dbsim*. In this case, a *x_rasgraf* process is started by *dbsim*. The process presents the updated runoff generation rate every time dbsim writes intermediate results.
- *RIBS_DISP_HYDRO* If this variable is defined, the on-line presentation of hydrographs is activated by *dbsim*. In this case, as many *x_rasgraf* processes as gauges are defined by the user are started by *dbsim*. Each process presents the updated hydrograph at a gauge site every time *dbsim* writes intermediate results.

Combining arguments in the command line and environmental variables, the user can operate the rainfall-runoff transformation module in different modes: with or without connection with the rainfall-generation module, with or without inclusion of forecasted rainfall loop and with or without graphic display of results.

For instance, the following shell script can be used to run the rainfall-runoff module off-line in simulation mode:

```
setenv  RIBS_T_GENE_START  0336:00
setenv  RIBS_T_BEGIN  0336:00
setenv  RIBS_T_GENE_END   0424:00
setenv  RIBS_DT_CALC  10.
setenv  RIBS_DT_RAIN  30.
setenv  RIBS_DT_RES  30.

setenv  RIBS_PATH_MEAS_RAIN  /model/data/st91/mrain
setenv  RIBS_PATH_FORE_RAIN  /model/data/st91/frain
setenv  RIBS_PATH_GEOMORF  /model/data/sieve/geomr
setenv  RIBS_PATH_STATES  /model/data/st91/stvar
setenv  RIBS_PATH_WATER_TABLE  /model/data/sieve/iniwt
setenv  RIBS_PATH_HYDROS  /model/data/st91/hydrg

setenv  RIBS_ROOT_GEOMORF  sieve
setenv  RIBS_ROOT_MEAS_RAIN  sieve
setenv  RIBS_ROOT_FORE_RAIN  sieve
setenv  RIBS_ROOT_HYDRO0  Fornacina
setenv  RIBS_ROOT_HYDRO1  BorgoSL
setenv  RIBS_ROOT_HYDRO2  Dicomano
setenv  RIBS_ROOT_HYDRO3  PVecchio
setenv  RIBS_ROOT_STATES  sieve
setenv  RIBS_ROOT_ORCALC  sieve
setenv  RIBS_ROOT_SOILDAT  sieve

setenv  RIBS_WATER_TABLE_FILE  47jan50ar

setenv  RIBS_EXT_POINTERS  ptr
setenv  RIBS_EXT_SOILS  soi
setenv  RIBS_EXT_STREAM_DIST  std
setenv  RIBS_EXT_SLOPES  slp
setenv  RIBS_EXT_WATER_TABLE  nwt
setenv  RIBS_EXT_STATE_NT  snt
setenv  RIBS_EXT_STATE_NF  snf
setenv  RIBS_EXT_STATE_MC  smc
setenv  RIBS_EXT_STATE_RF  srf
setenv  RIBS_EXT_HYDRO_MR  hydf
setenv  RIBS_EXT_MEAS_RAIN  mrf
setenv  RIBS_EXT_FORE_RAIN  frf
setenv  RIBS_EXT_ORCALC  ord
setenv  RIBS_EXT_SOILDAT  sdat
```

```
setenv  RIBS_PAR_EXP_DECAY  7.e-4
setenv  RIBS_PAR_ANIS_RATIO  500.
setenv  RIBS_PAR_RECHARGE_RATE  0.
setenv  RIBS_PAR_VEL_RATIO  12.75
setenv  RIBS_PAR_VEL_COEF  400.
setenv  RIBS_PAR_VEL_EXP  0.
setenv  RIBS_PAR_BASEFLOW  0.

setenv  RIBS_DISP_HYDRO  "/model/bin/x_hydgraf  -Q"

/model/bin/dbsim -A -R -G2 4 67 25 46
```

The option -A means that the model takes input files directly from their corresponding directory. The -F option is not selected, and therefore computations will only consider measured rainfall. The model would list all the files in the directory */model/data/st91/mrain* matching the template *sieve.hhhh:mm.mrf* and would use those with time tags between 0336:00 (initial time) and 0424:00 (end time) to compute basin evolution.

Since the -R option is activated, execution of this script generates a list of raster files in the directory */model/data/st91/stvar* containing the distribution of wetting front depth (extension *snf*), top front depth (extension *snt*), moisture content (extension *smc*) and runoff generation (extension *srf*). Files are also generated in the */model/data/st91/hydrg* directory. The -G option means that 2 gauges are selected for hydrograph generation, located at coordinates (4,67) and (25,46). File names for the first gauge follow the format *Fornacina.hhhh:mm.hyd*, and for the second gauge, *BorgoSL.hhhh:mm.hyd*.

The only display variable which is activated is *RIBS_DISP_HYDRO*, and therefore, only two windows would appear on the screen, each one corresponding to an *x_hydgraf* process showing hydrographs at the selected gauges. Other display processes can be activated defining the

corresponding variables. For instance, if the user wants to see the distribution of runoff generation, the following line must be added to the script:

```
setenv RIBS_DISP_RF "/model/bin/x_rasgraf -N25 -C/model/data/fabio.pal \
    -X 0.0001 10.0 -Q"
```

It activates the *x_rasgraf* process with the proper command line arguments.

Other variants of the off-line use are also straightforward. If the user wants also model evolution for forecasted rainfall, the last line of the script must be substituted by:

```
/model/bin/dbsim -A -F -G2 4 67 25 46
```

In this case the -R option is not necessary because model evolution with forecasted rainfall already implies writing intermediate results.

If the user is interested in hydrographs at other locations in the basin, the correct call is:

```
/model/bin/dbsim -A -R -G4 4 67 25 46 15 42 19 61
```

In this case, the gauges at Dicomano and Ponte Vecchio would also be added to model results.

To run the model on-line it must be used in connection with the rainfall generation module. The script file to use in this case must define the environmental variables required both by the rainfall generation module and by the rainfall-runoff transformation module. To activate the rainfall-runoff module in connection with the rainfall generation module the variable *RIBS_EXE_DBS* must be defined. *RIBS_EXE_DBS* contains the system call to run the rainfall-runoff module:

```
setenv RIBS_EXE_DBS "/model/bin/dbsim  -F -G2 4 67 25 46"
```

This declaration means that the model is run in forecasting mode generating hydrographs in two gauges: Fornacina and Borgo San Lorenzo. The option -A is now suppressed, since *dbsim* must read the names of input files from the standard input, as *rain_gen_gauge* generates them.

A3.4 The graphic display modules

There are three graphic display modules in RIBS: the raster viewer (*x_rasgraf*), the hydrograph viewer (*x_hydgraf*) and the pixel viewer (*x_pixgraf*). All of them share the same basic format and operation modes. They are programs to display files stored in the RIBS data base, and can be run without any additional data input. The raster viewer can, however, read another input file containing the description of the color palette. Programs read the name of the file to be displayed from the standard input. When the viewers are used in interactive mode, the user must type the file name at the computer terminal. When the viewers are used by a parent process, the parent must write the file name on the pipe communicating both processes.

The standard call to run display modules is:

% *program_name* [-Q] [*XWindows resources*] [*raster special options*]

The arguments in brackets are optional. The interpretation of the arguments is the following:

-*Q* : *Inhibit quit button*. When the viewers are started by a parent process, termination of the viewer using the quit button breaks the

pipeline between the parent and the child processes and may cause the parent to abort. To prevent that situation, the parent can call the viewer with the *-Q* option, which gives it total control about the termination of the process. To terminate the program in this case, the parent process should write *STOP* on the pipe.

X Window resource declaration: In standard X Window programs, resources which are not hardcoded can always be specified in the command line. The viewers also allow that possibility, because they pass the command line to *XtInitialize()* for interpretation. Standard X Window resources, such as window size or name, icon name, font type, etc., can therefore be included in the arguments to the viewers. Check the X Window reference for a complete listing of standard resource names and possible values.

Special options available in the raster viewer are the following:

-N : Number of divisions in the scale. The user may define the resolution used in the presentation. The format is *-Nn*, where *n* is an integer, indicating the number of divisions. If no value is specified, the number of divisions is taken from the default value specified in the raster file.

-C : Color palette file. The user may specify a color palette to use in the display. The format is *-Cfile_name*. This option is not compatible with a black and white display. If no color palette is specified, a default grayscale palette is used.

-X : Scale limits. The user may specify the upper and lower limits of the scale used by the raster display. The format is *-X low up*, where *low* is the lower limit and *up* is the upper limit. If values in the raster are higher than the upper limit or lower than the lower limit, special

colors are used to display them. If no scale limits are specified, upper and lower limits are set to the maximum and minimum values in the raster.

Examples of system calls to start the graphic display modules are available in the script files presented for the rainfall generation module and the rainfall-runoff transformation module. For instance, the declaration:

```
setenv RIBS_DISP_MEAS_RAIN "/model/bin/x_rasgraf -N25          \
    -C/model/data/fabio.pal -X 0.0001 10.0 -Q -n Measured_Rain"
```

contains the system call to start the raster viewer for measured rainfall presentation. The viewer is started with a scale of 25 divisions, ranging from 0.0001 to 10.0. Color code is taken from the palette described in *fabio.pal*. No quit button is offered to the user. Window name is set by the X Window resource -n, which is set to "*Measured_Rain*".

Interactive use

Interactive use is controlled by the window manager and the command buttons in the application. The window manager lets the user manage the application window. When the window is iconified, resized or restacked the application responds showing the image corresponding to the new conditions. The command buttons are used to transmit user requests to the application. The following command buttons are available in all viewers:

- *File button*: is used to specify a new file name. When the button is activated, a window pops up and prompts the user for the new file name. After the name is introduced, the new file is displayed.
- *Zoom button*: is used to zoom on the graphic image. When the button is activated, control is transferred to the user. The user clicks the mouse button in one of the corners of the new image and slides the mouse with the button pressed to the other corner. As the user slides the mouse, a rectangle is drawn, identifying the new image borders. When the mouse button is released, the new image is displayed. The raster viewer maintains the aspect ratio of the figure, and therefore the rectangle shown on the screen does not necessarily coincide with the position of the mouse, because it must keep the same aspect ratio as the display window.
- *Unzoom button*: is used to recover the original image. When the button is activated the image is displayed with its original boundaries.
- *Quit button*: is used to terminate the application. When the button is activated, the display window disappears.

There are also special command buttons in some viewers:
- *Include wt button*: is used in the pixel viewer to include or exclude the water table from view. The *Include wt* button is a toggle button, which alternates between the on and off position. When the button is *'off'*, the picture includes the soil column between the wetting front and the surface. When the button is *'on'*, the button is highlighted and the picture includes the soil column between the water table and the surface. It is a convenient option for pixels in which the water table is very deep.
- *Streamlines button*: is used by the pixel viewer to display a cross-sectional view of the pixel showing the flowpaths and front positions.

When the button is activated, a window pops up and presents the display. The popup window cannot be resized.

- *Pixel value button*: is used by the raster viewer to display the actual value corresponding to one pixel. When the button is activated, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted and the label widget shows the pixel's row, column and raster value. As the user slides the mouse with the button pressed, different pixels are highlighted and their values shown, until the button is released.

A3.5 The basin viewer module

The basin viewer, *x_bviewer*, is a program used to analyze model results produced by RIBS. It requires all the input files of *dbsim* plus the result files generated by a model run.

The standard call for the basin viewer is:

% x_bviewer *hhhh:mm* [*X Windows resources*] [-N] [-C]

The arguments in brackets are optional. The interpretation of the arguments is the following:

- *hhhh:mm* is the time tag of the basin state that is recovered. It must coincide with one of the intermediate basin states generated by the rainfall-runoff model.

X Windows : In standard X Window programs, resources which are not hardcoded can always be specified in the command line. The basin viewer also allows that possibility, because it passes the command line

to *XtInitialize()* for interpretation. Standard X Window resources, such as window size or name, icon name, font type, etc., can therefore be included in the arguments to the basin viewer. Check the X Window reference for a complete listing of standard resource names and possible values.

-*N : Number of divisions in the scale.* The user may define the resolution used by the basin viewer in the presentation of distributed variables. The format is -*Nn*, where *n* is an integer, indicating the number of divisions.

-*C : Color palette file.* The user may specify a color palette to use in the display of the basin viewer. The format is -*Cfile_name*. This option is not compatible with a black and white display. If no color palette is specified, a default grayscale palette is used.

The call to the basin viewer must be preceded by the declaration of all the required environmental variables. The basin viewer needs all the environmental variables of the rainfall-runoff module plus the display variables to present hydrographs and pixel states. The script for the basin viewer call should contain all the environmental variable declaration of the script presented in Section A3.3 plus the following:

```
setenv  RIBS_DISP_HYDRO  "/model/bin/x_hydgraf"
setenv  RIBS_DISP_PIXEL  "/model/bin/x_pixgraf"

/model/bin/x_bviewer 0365:00 -C/model/data/fablo.pal -N25 -n     \
Basin_State_Viewer
```

Note that the display module calls do not include the option -Q, since individual processes must be killed by the user. Upon execution of this

script, the basin viewer presents basin state at time 0365:00. The graphic display uses the color palette *fabio.pal*, with a scale of 25 units. The option -n is an X Window resource declaration, and its effect is naming the application window "*Basin_State_Viewer*".

Interactive use

Interactive use of *x_bviewer* is controlled by the window manager and the menus in the application. The window manager lets the user manage the application window. When the window is iconified, resized or restacked the application responds showing the image corresponding to the new conditions. The menus are used to transmit user requests to the application. The following pulldown menus are available in the application menu bar:

File menu

The *File* menu is used to interact with the database. It has two options:

- *Time* : is used to change the current time analyzed by the viewer. When the option is selected, a window pops up containing all the time tags available for the current model run. The user selects the required time by clicking on one of the items displayed and activating the *OK* button. After the time tag is selected, the basin viewer displays the same variable for the new time reference.
- *Quit* : is used to terminate the application. When the option is selected, the display window disappears. The application does not close the open

pipes to children processes, which should be killed independently by the user.

Edit menu

The *Edit* menu is used to configure the display. It has two options:

- *Zoom* : is used to zoom on the graphic image. When the option is selected, control is transferred to the user. The user clicks the mouse button in one of the corners of the new image and slides the mouse with the button pressed to the other corner. As the user slides the mouse, a rectangle is drawn, identifying the new image borders. When the mouse button is released, the new image is displayed. The basin viewer maintains the aspect ratio of the figure, and therefore the rectangle shown on the screen does not necessarily coincide with the mouse position, because it must keep the same aspect ratio as the display window.
- *Unzoom* : is used to recover the original image. When the option is selected the image is displayed with its original boundaries.

Variable menu

The *Variable* menu is used to select the virtual variable which should be displayed. When the application starts, it displays the wetting front depth. There are eleven options:

- *Wetting front*: is used to display the wetting front position. When the option is selected the variable is displayed.

- *Top front*: is used to display the top front position. When the option is selected the variable is displayed.
- *Moisture content*: is used to display the moisture content in the pixel. When the option is selected the variable is displayed.
- *Runoff generation*: is used to display the runoff generation. When the option is selected the variable is displayed.
- *Rainfall*: is used to display the rainfall rate. When the option is selected the variable is displayed.
- *Hydraulic conductivity*: is used to display the surface normal hydraulic conductivity. When the option is selected the variable is displayed.
- *Infiltration capacity*: is used to display the pixel maximum infiltration capacity. When the option is selected the variable is displayed.
- *Flow at front*: is used to display the normal flow at the level of the wetting front. When the option is selected the variable is displayed.
- *Upper deficit*: is used to display the moisture deficit above the wetting front. When the option is selected the variable is displayed.
- *Upper saturation*: is used to display the moisture deficit above the wetting front (% with respect to saturation). When the option is selected the variable is displayed.
- *Lower deficit*: is used to display the moisture deficit between the wetting front and the. When the option is selected the variable is displayed.
- *Total deficit*: is used to display the total moisture deficit in the column. When the option is selected the variable is displayed.
- *Total saturation*: is used to display the moisture deficit in the soil column (% with respect to saturation). When the option is selected the variable is displayed.

- *Distance to stream*: is used to display the distance to nearest stream. When the option is selected the variable is displayed.

Hydrograph menu

The *Hydrograph* menu is used to obtain different types of hydrographs. Three options are available:

- *Local hydrograph*: is used to generate the hydrograph at a point in the basin. The hydrograph is generated from the beginning of the storm up to the current time. When the option is selected, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted. As the user slides the mouse with the button pressed, different pixels are highlighted and their row and column indices shown in the label widget, until the button is released. When the button is released, the viewer creates a gauge at the selected location, recovers all previous basin states and generates the hydrograph, writing results to a file. The format of the file name is *irnjcn.hhhh:mm.ext*, where *rn* is the row number, *cn* is the column number, *hhhh:mm* is the current time tag and *ext* is the extension, taken from the environmental variable *RIBS_EXT_HYDROS* . *x_bviewer* then starts a *x_hydgraf* process to display the results. The child process pops up an independent window, which is controlled by the user.

- *Pixel hydrograph*: is used to generate the hydrograph generated by a single pixel in the basin. The hydrograph is generated from the beginning of the storm up to the current time. When the option is selected, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted. As the user slides the mouse with the button

pressed, different pixels are highlighted and their row and column indices shown in the label widget, until the button is released. When the button is released, the viewer creates a gauge at the selected location, recovers all previous basin states and generates the hydrograph, writing results to a file. The format of the file name is *irnjcn.hhhh:mm.phd*, where *rn* is the row number, *cn* is the column number, *hhhh:mm* is the current time tag and *phd* is the extension. *x_bviewer* then starts a *x_hydgraf* process to display the results. The child process pops up an independent window, which is controlled by the user.

- *Unit hydrograph*: is used to generate the unit basin response function at a point in the basin. The unit hydrograph is obtained for basin state at the current time. When the option is selected, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted. As the user slides the mouse with the button pressed, different pixels are highlighted and their row and column indices shown in the label widget, until the button is released. When the button is released, the viewer creates a gauge at the selected location, evaluates the infiltration-excess runoff for unit rainfall upstream the gauge and generates the hydrograph, writing results to a file. The format of the file name is *irnjcn.hhhh:mm.uhy*, where *rn* is the row number, *cn* is the column number, *hhhh:mm* is the current time tag and *uhy* is the extension. *x_bviewer* then starts a *x_hydgraf* process to display the results. The child process pops up an independent window, which is controlled by the user.

Pixel menu

The *Pixel* menu is used to obtain information about an individual pixel. Three options are available:

- *Variable report*: is used to generate a report of the time evolution of the currently active virtual variable for a pixel. The report is generated from the beginning of the storm up to the current time. When the option is selected, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted. As the user slides the mouse with the button pressed, different pixels are highlighted and their row and column indices shown in the label widget, until the button is released. When the button is released, the viewer recovers all previous basin states and generates the time evolution of the variable, writing results to a *hydrograph* file. The format of the file name is *irnjcn.hhhh:mm.rep*, where *rn* is the row number, *cn* is the column number, *hhhh:mm* is the current time tag and *rep* is the extension. *x_bviewer* then starts a *x_hydgraf* process to display the results. The child process pops up an independent window, which is controlled by the user.
- *Pixel value*: is used to display the actual value of the active virtual variable corresponding to a pixel. When the button is activated, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted and the label widget shows the pixel's row, column and raster value. As the user slides the mouse with the button pressed, different pixels are highlighted and their values shown, until the button is released.

- *Pixel display* is used to present a graphic display showing the moisture profile of the pixel. When the option is selected, control is transferred to the user. When the user clicks on a pixel, the pixel rectangle is highlighted. As the user slides the mouse with the button pressed, different pixels are highlighted and their row and column indices shown in the label widget, until the button is released. When the button is released, the viewer generates a *pixel* file containing all variables needed to determine the state of that pixel. The format of the file name is *irnjcn.hhhh:mm.pxl*, where *rn* is the row number, *cn* is the column number, *hhhh:mm* is the current time tag and *pxl* is the extension. *x_bviewer* then starts a *x_pixgraf* process to display the results. The child process pops up an independent window, which is controlled by the user.

A3.6 Environmental variables

This section presents the environmental variables used by the RIBS package. Name, format and meaning are provided for every variable.

General simulation parameters

Name: RIBS_T_GENE_START
Format: hhhh:mm (hours since the beginning of the year)
Meaning: Generic time start. Corresponds to the time reference since the beginning of the storm.

Name: RIBS_T_GENE_END

Format: hhhh:mm (hours since the beginning of the year)

Meaning: Generic time end. End of model computations.

Name: RIBS_T_BEGIN

Format: hhhh:mm (hours since the beginning of the year)

Meaning: Starting time of model computations. May not correspond to the generic time start if the model is initialized at some intermediate state.

Name: RIBS_DT_CALC

Format: Floating point value (minutes)

Meaning: Computation time increment.

Name: RIBS_DT_RAIN

Format: Floating point value (minutes)

Meaning: Duration of the forecasted rainfall. The duration of the measured rainfall is inferred from the time tag of the previous file.

Name: RIBS_DT_RES

Format: Floating point value (minutes)

Meaning: Time step in the presentation of model results.

Path variables

Name: RIBS_PATH_MEAS_RAIN

Format: String of less than 80 characters

Meaning: Path for measured rainfall files

Name: RIBS_PATH_FORE_RAIN

Format: String of less than 80 characters

Meaning: Path for forecasted rainfall files

Name: RIBS_PATH_HIST_RAIN

Format: String of less than 80 characters

Meaning: Path for hyetograph files

Name: RIBS_PATH_GEOMORF

Format: String of less than 80 characters

Meaning: Path for files containing the geomorphologic and pedologic description of the basin.

Name: RIBS_PATH_STATES

Format: String of less than 80 characters

Meaning: Path for state variable files

Name: RIBS_PATH_HYDROS

Format: String of less than 80 characters

Meaning: Path for hydrograph files

Root file names

Name: RIBS_ROOT_GEOMORF

Format: String of less than 80 characters

Meaning: Root name of geomorphology files

Name: RIBS_ROOT_MEAS_RAIN

Format: String of less than 80 characters

Meaning: Root name of measured rain files

Name: RIBS_ROOT_FORE_RAIN

Format: String of less than 80 characters

Meaning: Root name of forecasted rain files.

Name: RIBS_ROOT_HIST_RAIN

Format: String of less than 80 characters

Meaning: Root name of hyetograph files.

Name: RIBS_ROOT_HYDROn, where n is a non-negative integer

Format: String of less than 80 characters

Meaning: Root name of hydrograph files corresponding to gauge n.

Name: RIBS_ROOT_STATES

Format: String of less than 80 characters

Meaning: Root name of state variable files.

Name: RIBS_ROOT_ORCALC

Format: String of less than 80 characters

Meaning: Root name of file containing the order of computation.

Name: RIBS_ROOT_SOILDAT

Format: String of less than 80 characters

Meaning: Root name of file containing the soil types.

Name: RIBS_WATER_TABLE_FILE

Format: string of less than 80 characters

Meaning: File name of the initial water table position

Extension file names

Name: RIBS_EXT_POINTERS

Format: String of less than 10 characters

Meaning: Extension for the file containing the basin pointers.

Name: RIBS_EXT_SOILS

Format: String of less than 10 characters

Meaning: Extension for the file containing the basin soil classes.

Name: RIBS_EXT_STREAM_DIST

Format: String of less than 10 characters

Meaning: Extension for the file containing the distance to the nearest stream.

Name: RIBS_EXT_GAUGE_DIST

Format: String of less than 10 characters

Meaning: Extension for the file containing the distance to basin outlet along stream channels.

Name: RIBS_EXT_SLOPES

Format: String of less than 10 characters

Meaning: Extension for the file containing the basin slopes.

Name: RIBS_EXT_WATER_TABLE

Format: String of less than 10 characters

Meaning: Extension for the file containing the initial water table depth.

Name: RIBS_EXT_STATE_NT

Format: String of less than 10 characters

Meaning: Extension for the files containing the top front depth.

Name: RIBS_EXT_STATE_NF

Format: String of less than 10 characters

Meaning: Extension for the files containing the wetting front depth

Name: RIBS_EXT_STATE_MC

Format: String of less than 10 characters

Meaning: Extension for the moisture content files.

Name: RIBS_EXT_STATE_RF

Format: String of less than 10 characters

Meaning: Extension for the files containing the runoff generation rate.

Name: RIBS_EXT_HYDRO_MR

Format: String of less than 10 characters

Meaning: Extension for the hydrograph files.

Name: RIBS_EXT_MEAS_RAIN

Format: String of less than 10 characters

Meaning: Extension for the measured rainfall files.

Name: RIBS_EXT_FORE_RAIN

Format: String of less than 10 characters

Meaning: Extension for the forecasted rainfall files.

Name: RIBS_EXT_HIST_RAIN

Format: String of less than 10 characters

Meaning: Extension for the hyetograph files.

Name: RIBS_EXT_ORCALC

Format: String of less than 10 characters

Meaning: Extension for the file containing the basin pointers

Name: RIBS_EXT_SOILDAT

Format: String of less than 10 characters

Meaning: Extension for the file containing the basin pointers

Model parameters

Name: RIBS_PAR_RECHARGE_RATE

Format: Floating point value (mm/h)

Meaning: Recharge rate in the basin. If set to 0.0, recharge rate is obtained considering saturation at the water table level.

Name: RIBS_PAR_EXP_DECAY

Format: Floating point value (mm^{-1})

Meaning: Parameter f: length scale controlling the exponential decay of hydraulic conductivity with depth.

Name: RIBS_PAR_ANIS_RATIO

Format: Floating point value

Meaning: Anisotropy ratio of hydraulic conductivity.

Name: RIBS_PAR_VEL_RATIO

Format: Floating point value

Meaning: Ratio of channel to hillslope velocity.

Name: RIBS_PAR_VEL_COEF

Format: Floating point value (m/h)

Meaning: Coefficient of the power law relating hillslope velocity and discharge at the outlet.

Name: RIBS_PAR_VEL_EXP

Format: Floating point value

Meaning: Exponent of the power law relating hillslope velocity and discharge at the outlet.

Name: RIBS_PAR_BASEFLOW

Format: Floating point value (m^3/s)

Meaning: Baseflow at the basin final outlet.

System calls

Name: RIBS_EXE_DBS

Format: System call in one single line

Meaning: System call to start the rainfall-runoff module from the rainfall acquisition module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_MEAS_RAIN

Format: System call in one single line

Meaning: System call to start the process to display measured rainfall from the rainfall acquisition module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_FORE_RAIN

Format: System call in one single line

Meaning: System call to start the process to display forecasted rainfall from the rainfall acquisition module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_MONI_RAIN

Format: System call in one single line

Meaning: System call to start the process to display hyetographs from the rainfall acquisition module. It contains the full pathname of the

executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_NF

Format: System call in one single line

Meaning: System call to start the process to display wetting front depth from the rainfall-runoff module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_NT

Format: System call in one single line

Meaning: System call to start the process to display top front depth from the rainfall-runoff module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_MC

Format: System call in one single line

Meaning: System call to start the process to display moisture content from the rainfall-runoff module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_RF

Format: System call in one single line

Meaning: System call to start the process to display runoff generation rate from the rainfall-runoff module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

Name: RIBS_DISP_HYDRO

Format: System call in one single line

Meaning: System call to start the process to display hydrographs from the rainfall-runoff module. It contains the full pathname of the executable file and the proper arguments. If the variable is not defined, the process is not started.

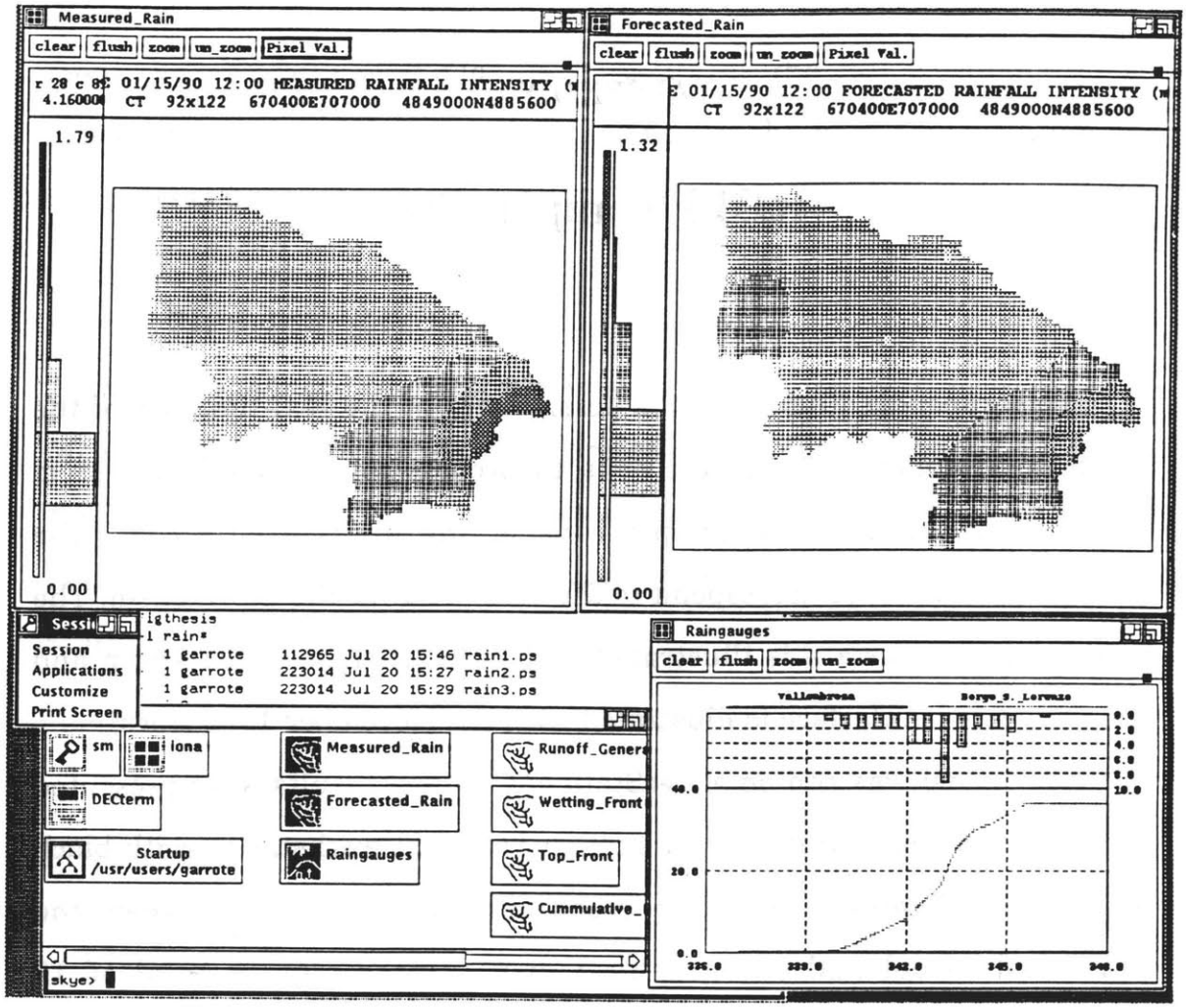
APPENDIX 4

RIBS Sample Run

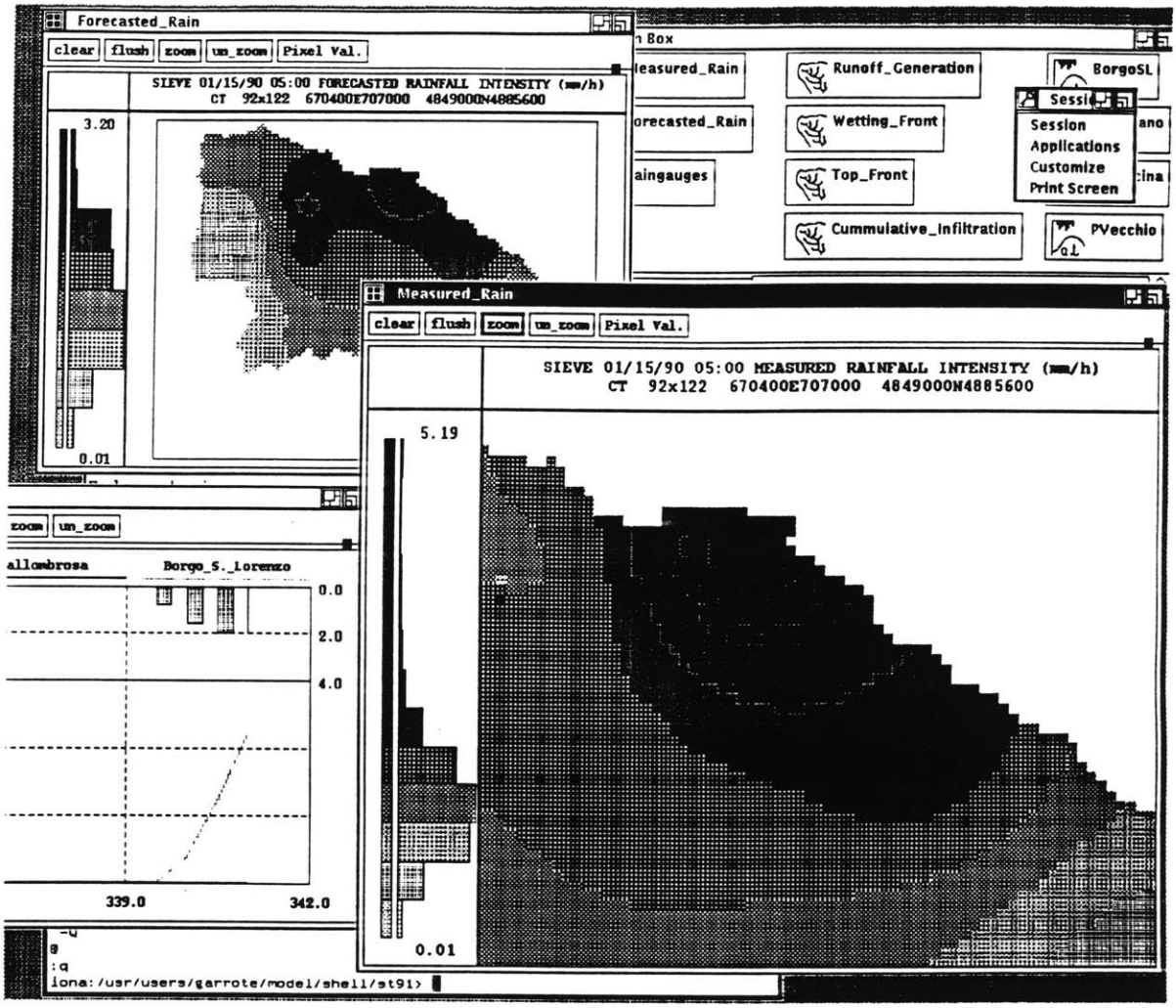
This appendix illustrates the possibilities of the real-time use of the RIBS environment. The objective is to provide an overview of system capabilities through two sample runs of the distributed model. The examples shown here correspond to the case study of the Sieve basin. The model-driven interface is illustrated with the November 1991 storm and the user-driven interface is illustrated with the November 1982 storm.

The model was run on a DecStation 3100 and results were presented through a computer network on another identical workstation with black and white display. Computer resources were shared between the numerical model and the graphical display processes. A total of 12 different processes were simultaneously active on the remote host. Rainfall data for the November 1991 storm had a temporal resolution of 30 min. The model was run with computation time steps of 10 min in forecasting mode. In these conditions, the model updates results for a full forecasting cycle in less than 1 min, with acceptable performance in the graphic operations.

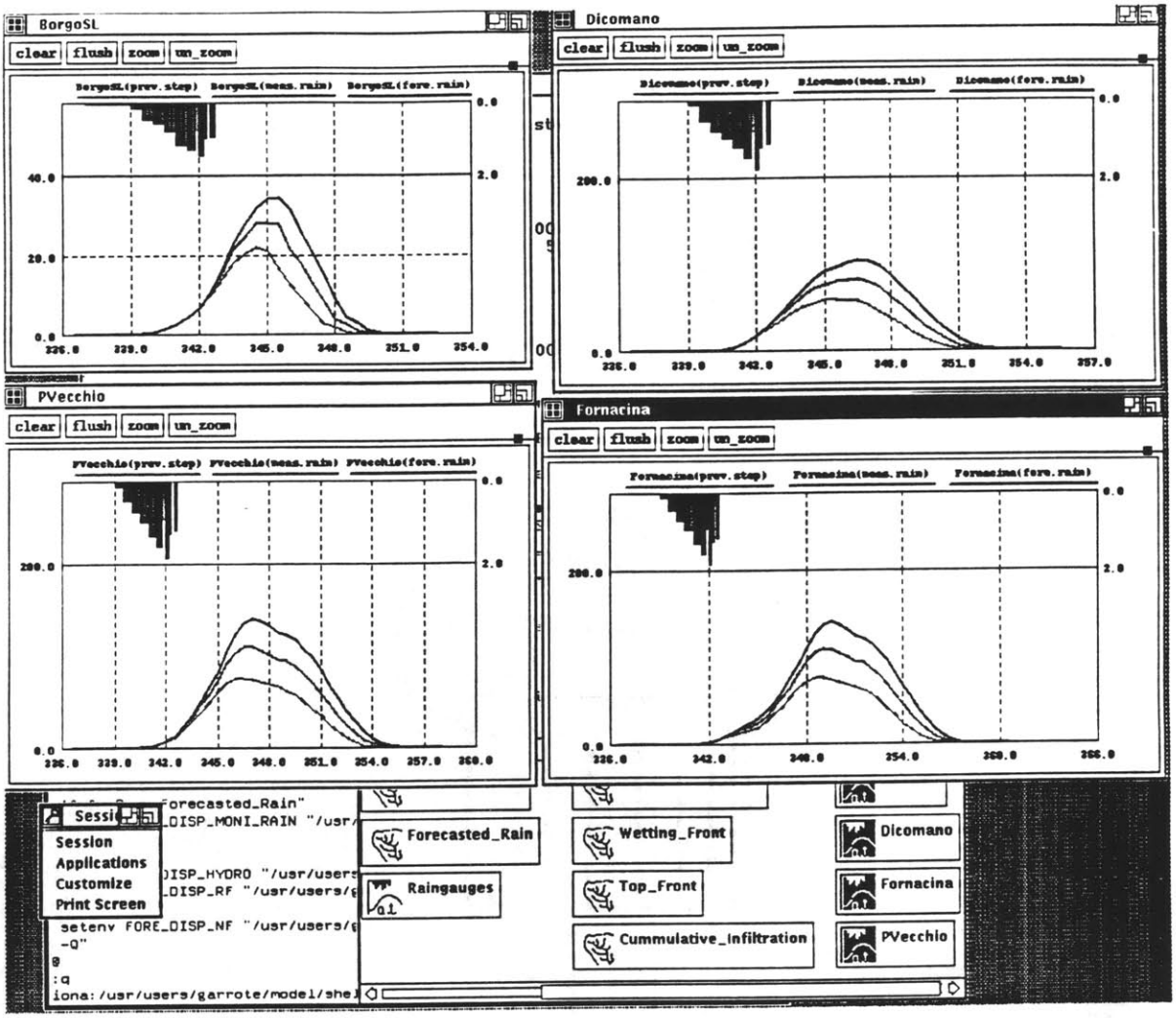
The appendix consists of a series of reproductions of the computer screen as the model is running. The first 8 figures correspond to the model-driven interface, and the remaining 5 figures correspond to the user-driven interface.



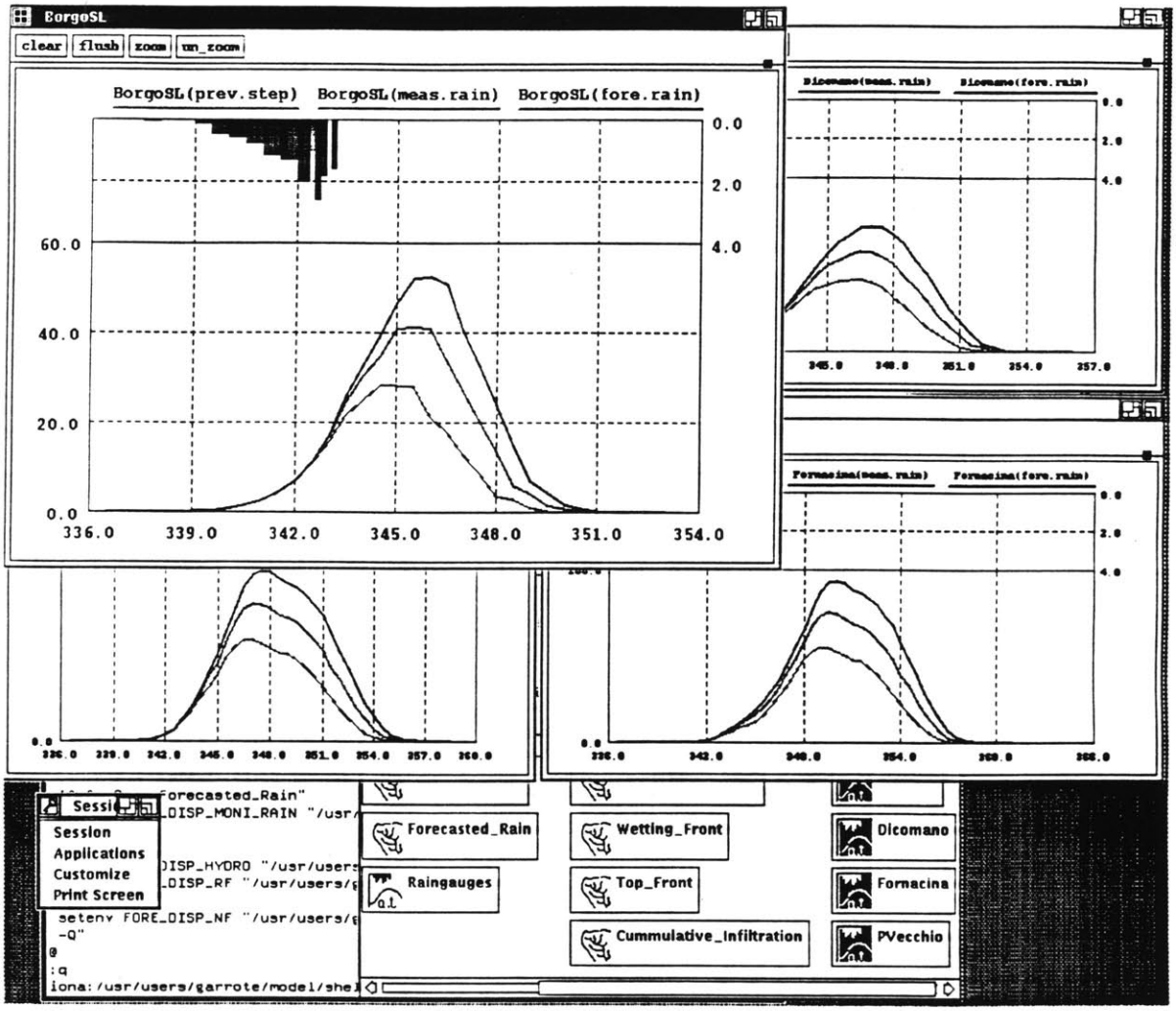
This figure presents the basic rainfall information offered by the model-driven interface of RIBS. Two raster viewer processes present rainfall distribution in the basin. The window on the upper left corner presents measured rainfall and the window on the upper right corner presents forecasted rainfall. The window on the lower right corner is a hydrograph viewer which presents hyetographs of two raingauges in the basin: Vallombrosa and Borgo San Lorenzo. The upper plot in this window presents rainfall intensity and the lower plot presents cumulative rainfall. The lower left corner shows the icon box, where other RIBS windows can be accessed.



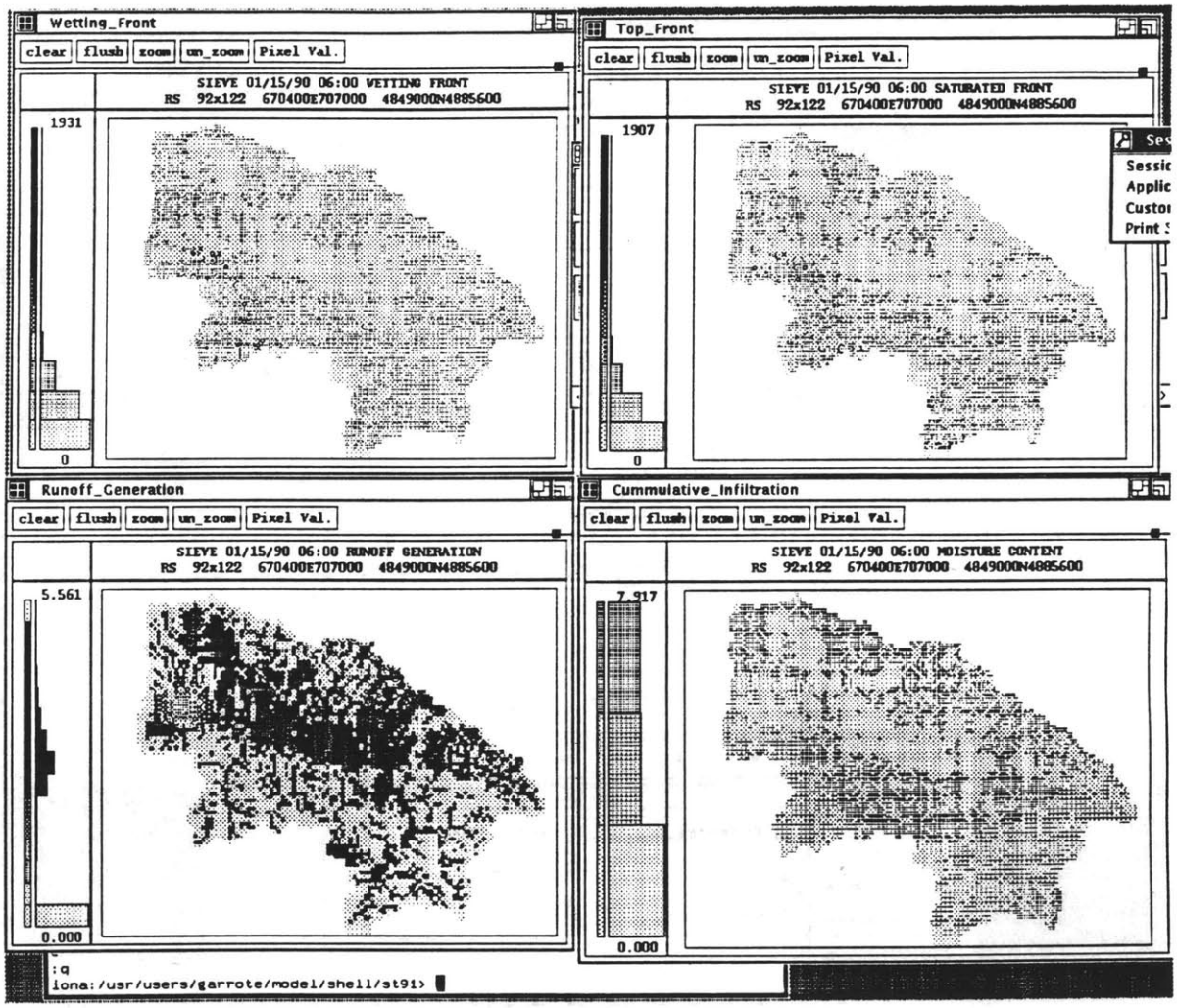
Windows in the RIBS environment are directly managed by the user. The position, stacking order and size can be changed with simple mouse operations. This image presents another screen configuration for rainfall information. The measured rainfall window has been enlarged, and the image is zoomed on the central part of the basin, where rainfall intensity is maximum. The forecasted rain window and the hyetograph window remain on the background. These and all other windows managed by the model-driven interface are automatically updated every time the model generates new results. The user can rearrange window positions and configurations, but window contents at any time are controlled by RIBS.



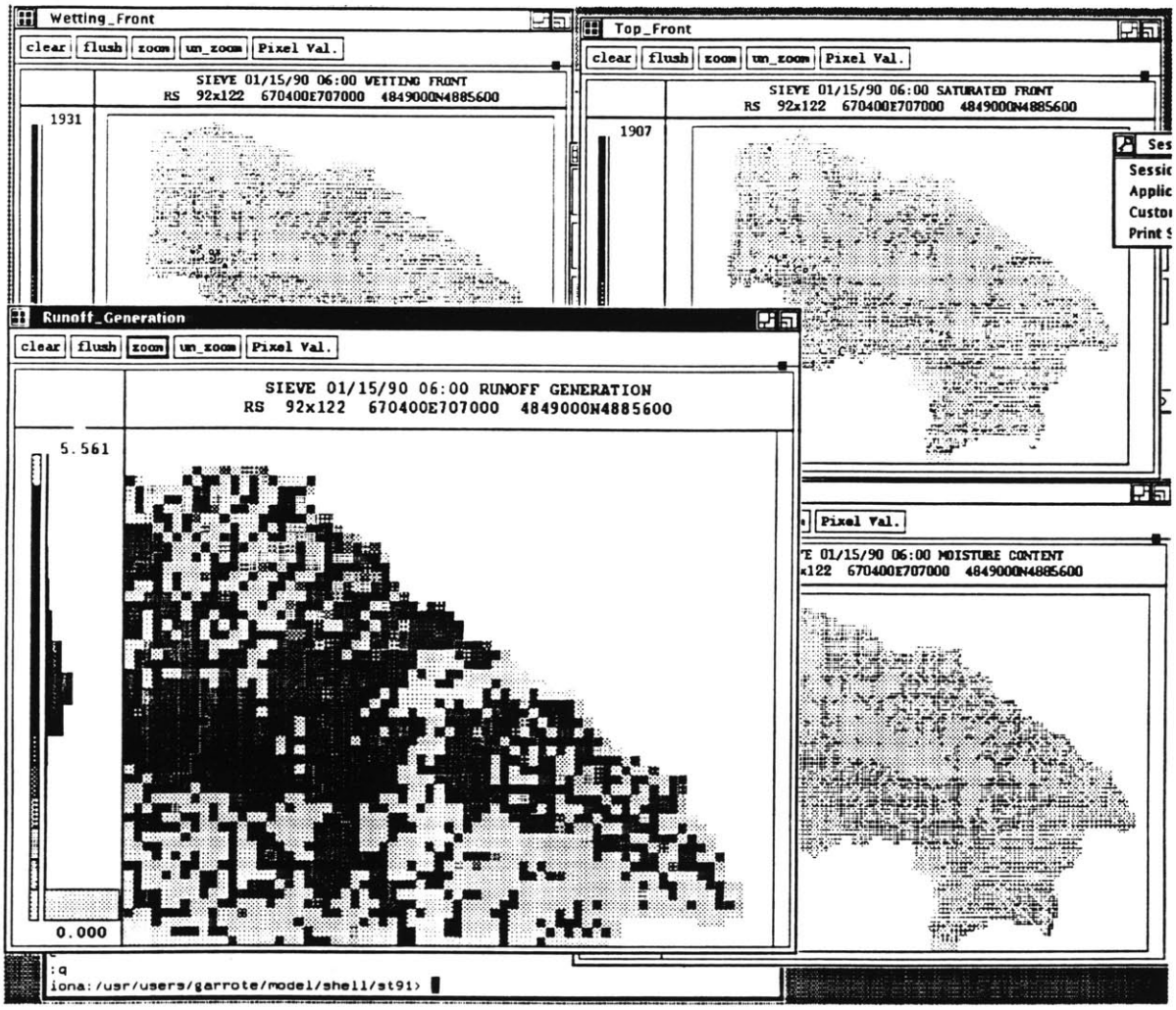
This figure presents the basic streamflow information offered by the model. Model results for the four active gauges are simultaneously shown in the screen. The upper left is Borgo San Lorenzo, the upper right is Dicomano, the lower left is Ponte Vecchio and the lower right is Fornacina, the final outlet. Since the model is running in forecasting mode, each hydrograph viewer presents three time series of rainfall and streamflow: model predictions in the last time step, model predictions considering only measured rainfall and model predictions considering measured and forecasted rainfall.



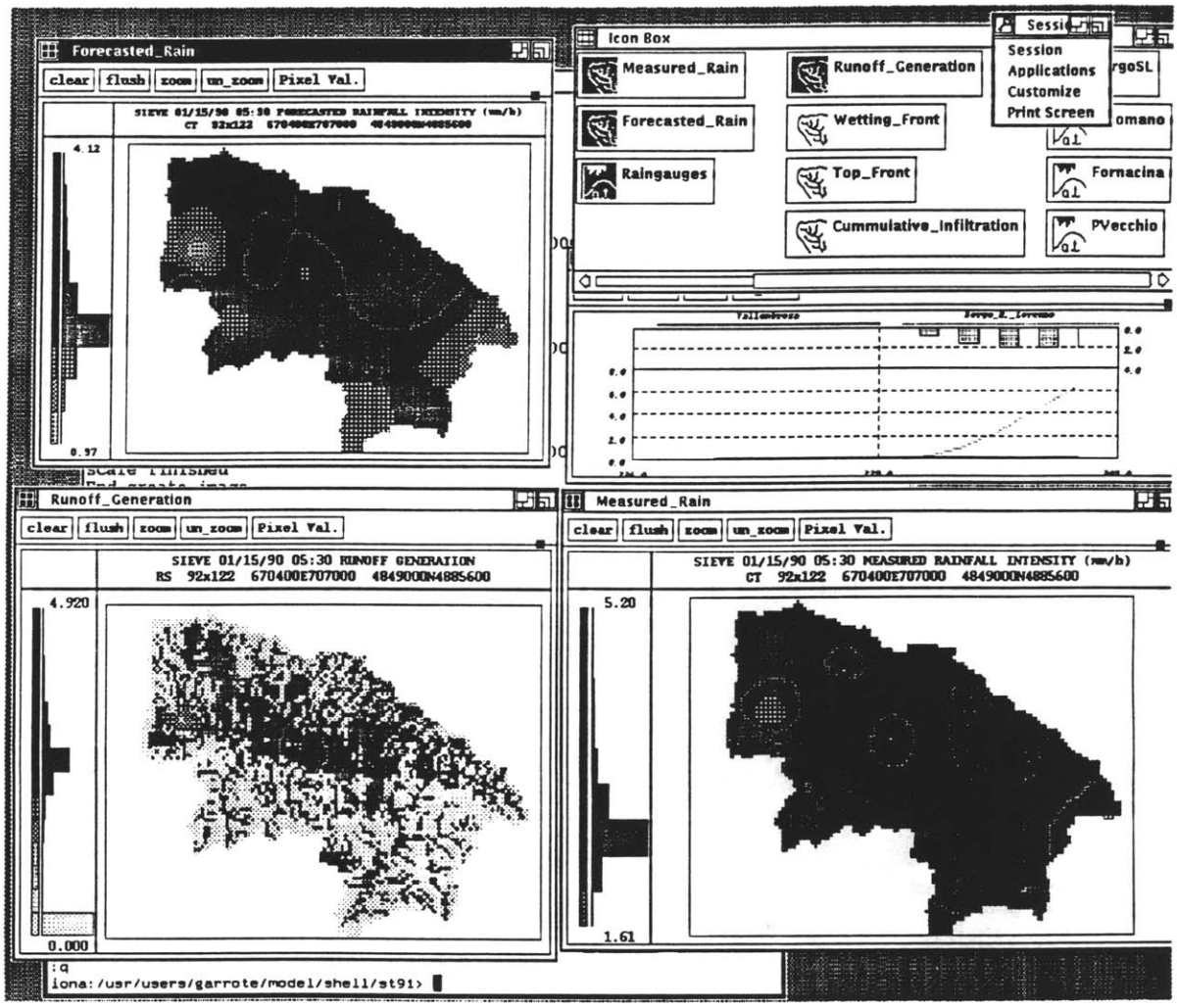
This picture shows the Borgo San Lorenzo window enlarged for a more detailed analysis. The three hydrographs provided by the model offer a dynamic picture of the situation at that location. Recent and future changes can be analyzed comparing model results for the previous, current and future time steps. Local information at that point can be compared with information obtained at other locations in the basin, to analyze the spatially varying effects of rainfall.



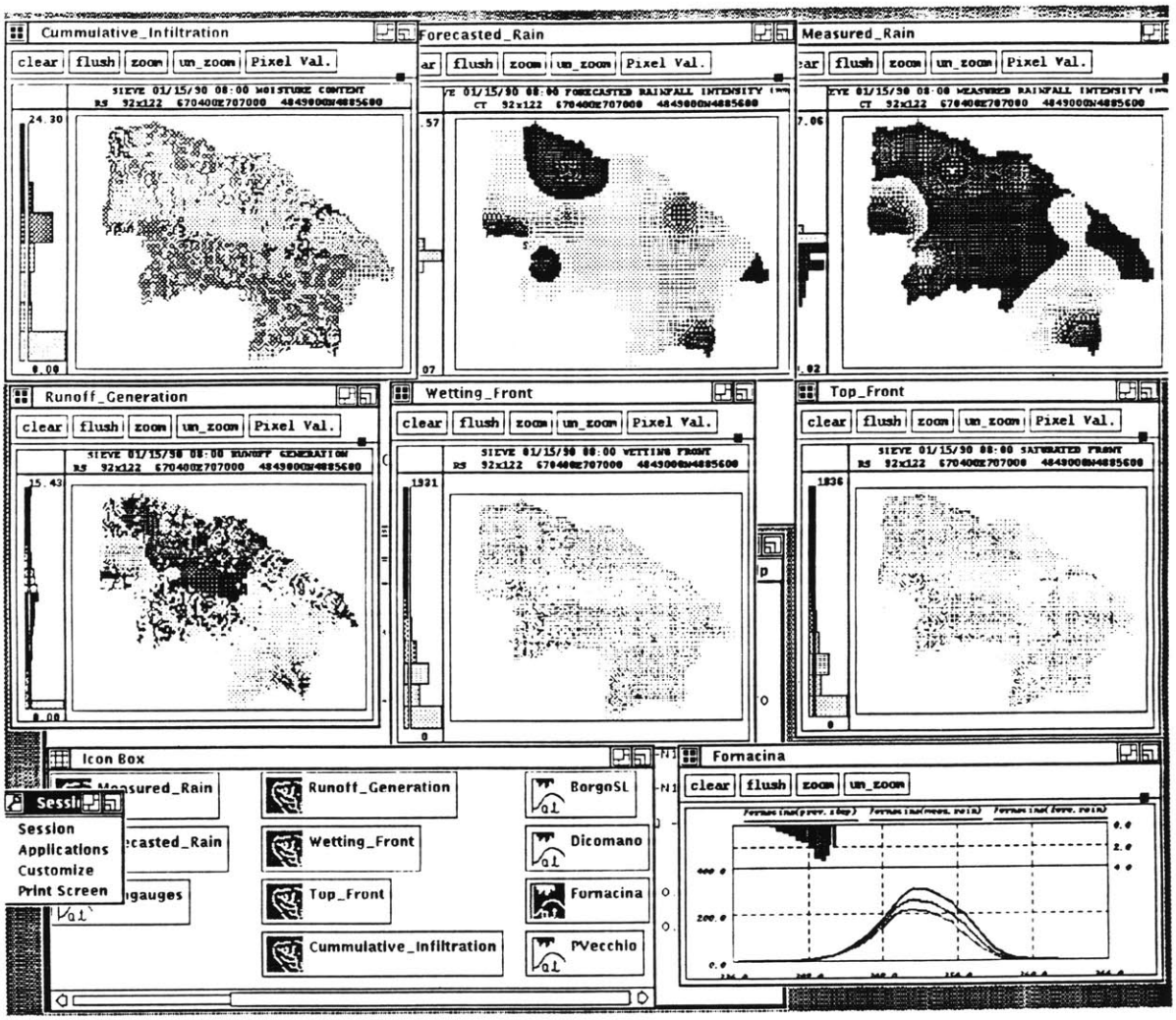
This image presents the basin state information offered by the model-driven interface of RIBS. The window on the upper left corner presents the spatial distribution of wetting front depths. The upper right corner corresponds to top front depths. The third state variable, moisture content, is shown on the lower right corner. The model also offers information about the spatial distribution of runoff generation in the basin, presented here in the window on the lower left corner.



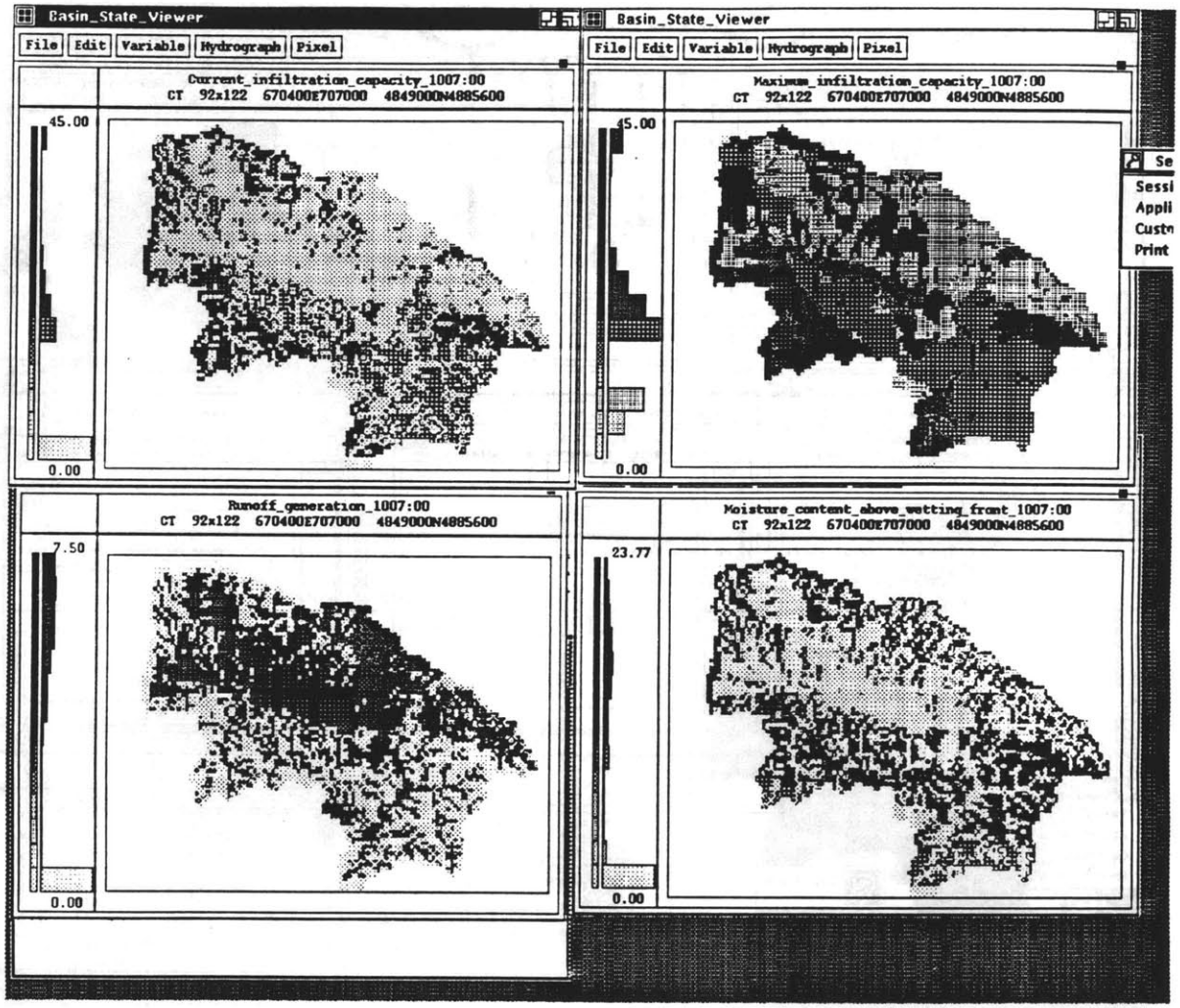
As in previous cases, these windows can be reconfigured by the user to analyze particular aspects. This image shows the runoff-generation window enlarged and focused on the northeastern area of the Sieve basin, where soils have lower permeability. This area shows many pixels in the surface-saturated state, which are generating runoff while the rainfall intensity is lower than their surface hydraulic conductivity.



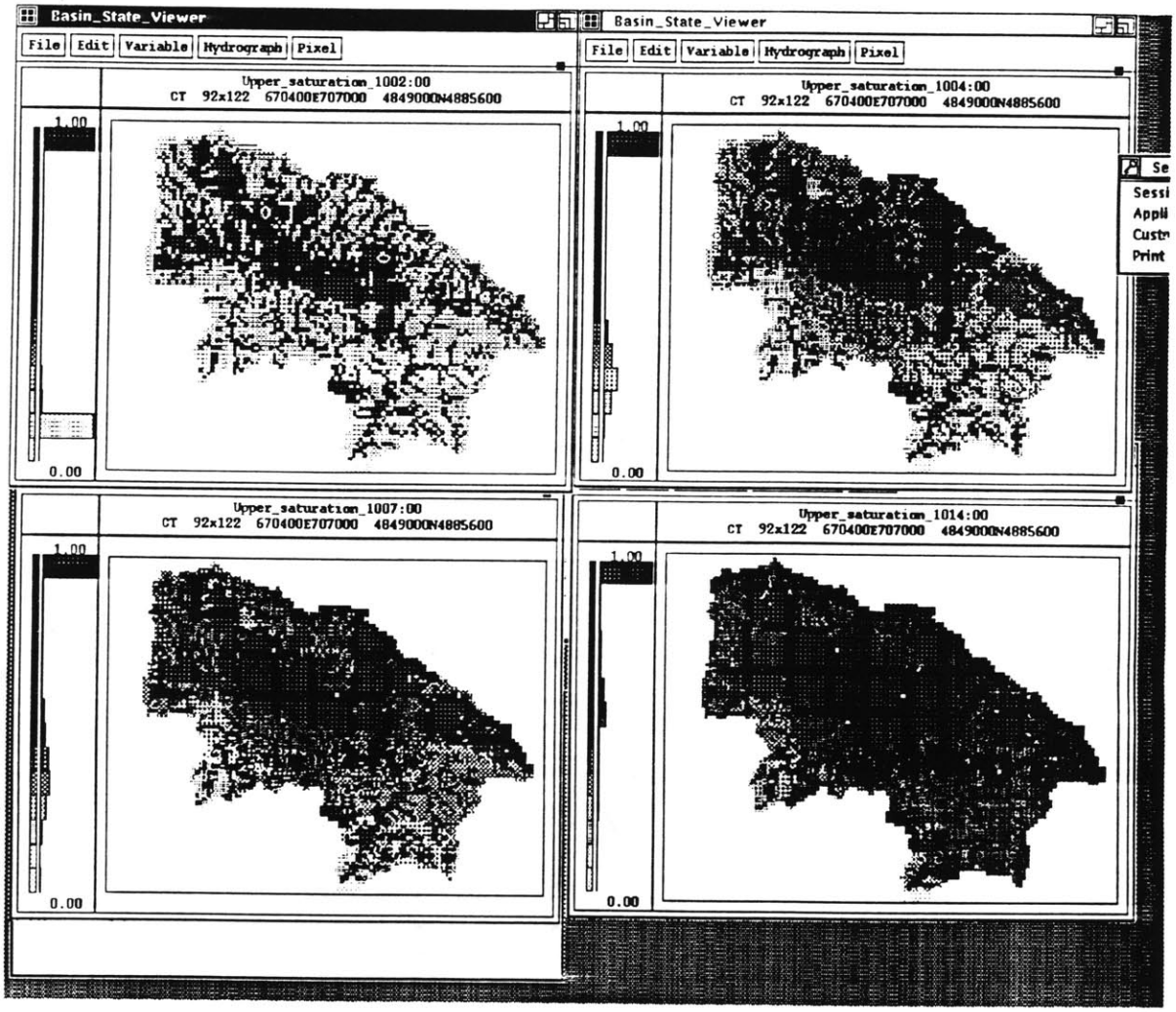
The eleven result windows presented so far can be combined in any way. Icons of all windows with their corresponding names are located in the icon box, which is shown on the upper right corner of the figure. Icons of magnified windows appear highlighted, while icons of non-active windows appear dimmed. The user can magnify any window by selecting its icon in the icon box. The configuration shown in the figure presents measured and forecasted rainfall compared with runoff generation in the basin.



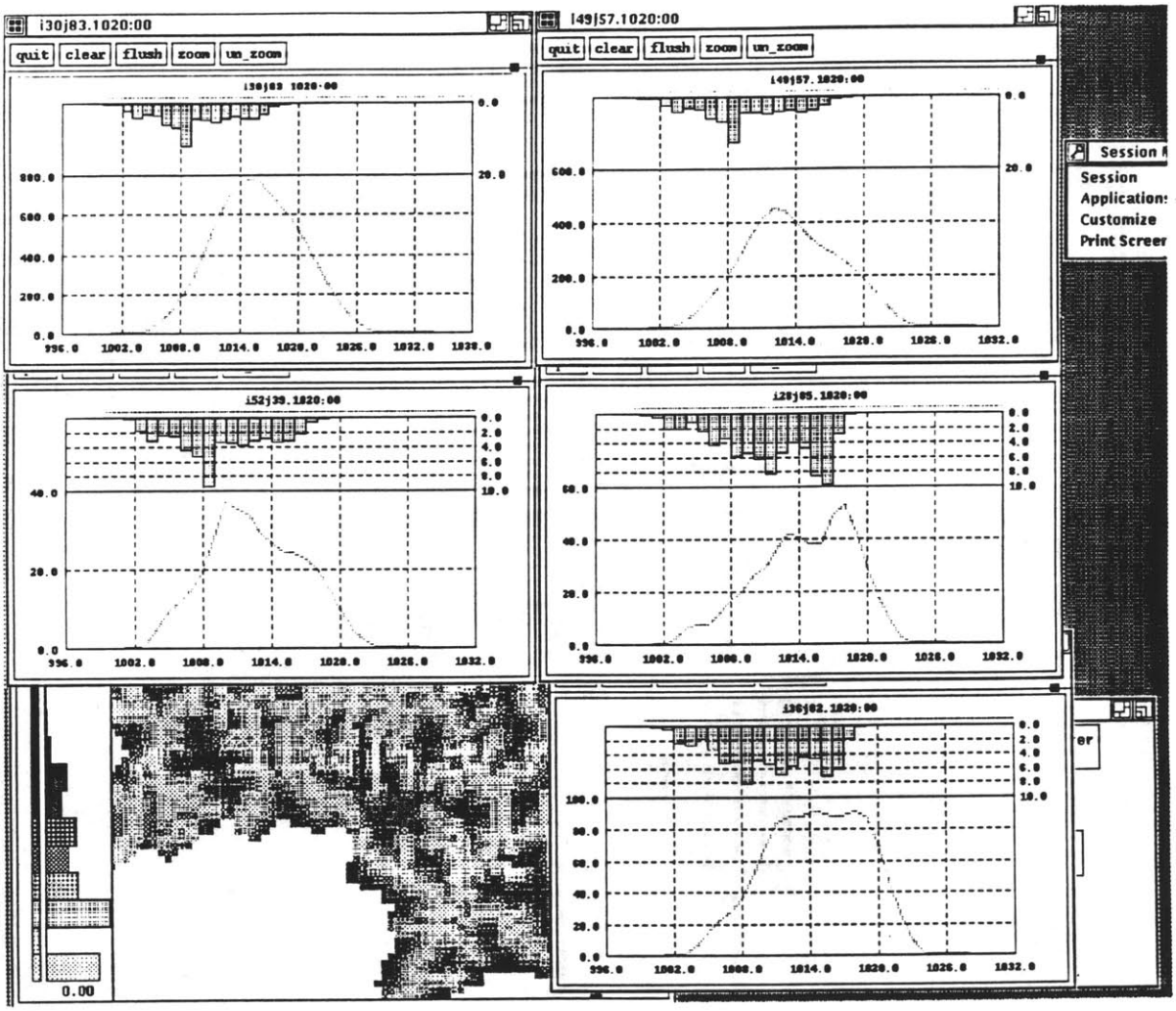
The user can magnify as many windows as he or she wishes. This image shows the six raster viewers and the hydrograph at the outlet, together with the icon box. Zooming and resizing capabilities are simultaneously active for all processes. The basic idea of this interface style is to offer large amounts of information to the user in a flexible way, so that he or she can dynamically select which aspects to consult according to the changing situation.



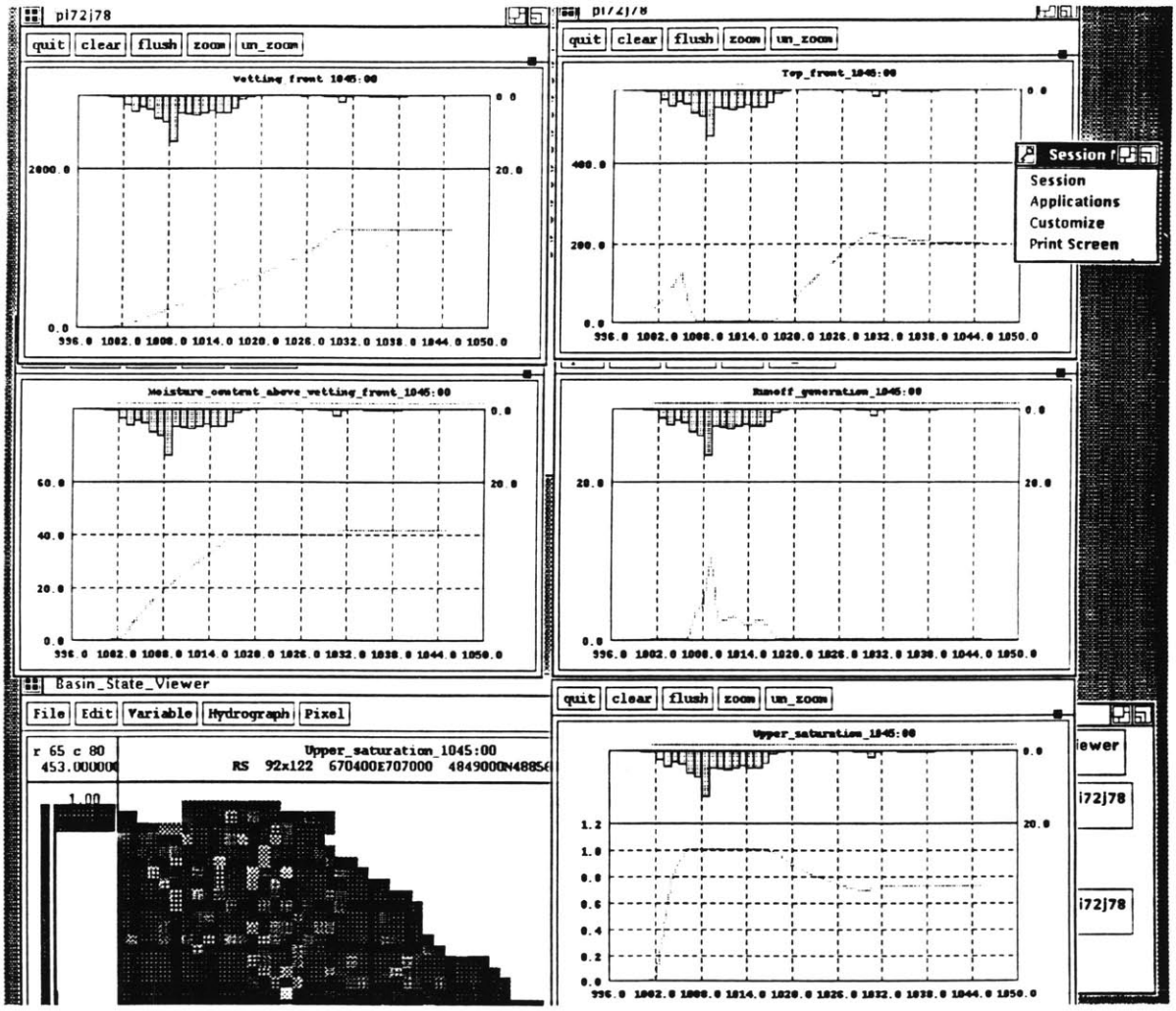
This is the first image corresponding to the user-driven interface, which can be used to access the results and states stored in the database. The user can simultaneously run several basin viewers to consult different aspects of model evolution. In this example, four different viewers present four variables at the same stage of model evolution. The window on the upper left corner presents the current infiltration capacity, showing the extent of saturated areas. The window on the upper right corner presents the surface hydraulic conductivity for comparison. Model state variables, such as moisture content (lower right) or results, such as runoff generation (lower left) are also available in the basin viewers.



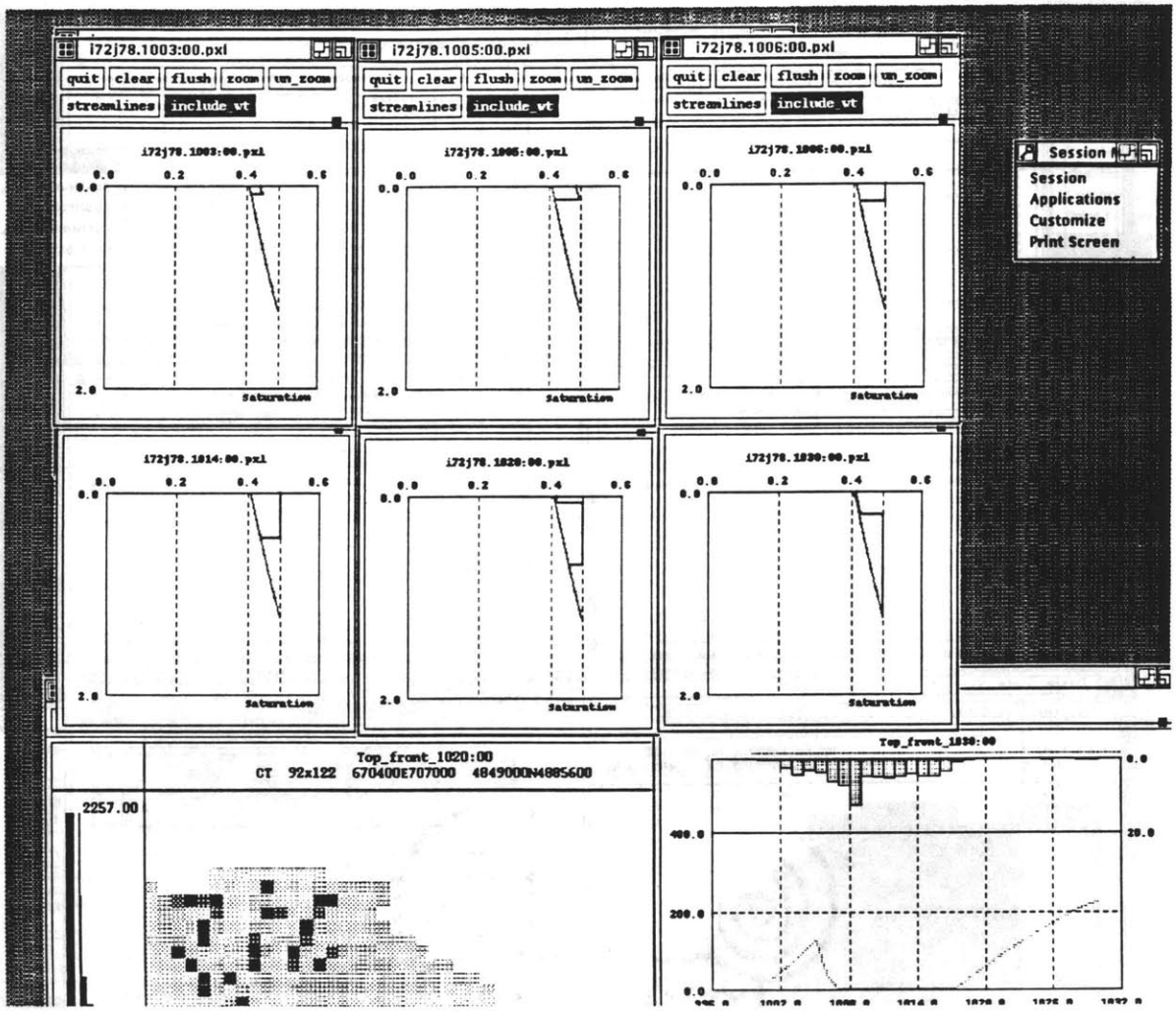
Unlike the processes of the model-driven interface, the basin viewer allows the user to set the time at which results are to be consulted. This feature can be used, for instance, to analyze the time evolution of distributed variables. This example shows the spatial distribution of the degree of saturation above the wetting front at four different times. The upper left window corresponds to 1002:00, the upper right to 1004:00, the lower left to 1007:00 and the lower right to 1014:00. The dynamic nature of the saturated areas in the basin during the storm is apparent in this image.



This image illustrates the possibilities offered by the *Hydrograph* menu of the basin viewer. The option *Local Hydrograph* has been used to generate hydrographs at five different locations in the basin, which are identified by the coordinates of the pixel and the time tag. The points were interactively selected by the user, by clicking on the mouse at the desired location. The basin viewer responds creating a new process which presents the requested hydrograph. That process appears on an independent window which can be directly managed by the user. The response time is of the order of a few seconds, and depends on how many previous model states have to be consulted to generate the hydrographs.



The basin viewer also offers the possibility of generating reports of the time evolution of virtual variables for an individual pixel. This image shows the time evolution of the wetting and top front depths, moisture content, local runoff generation and degree of saturation above the wetting front for the pixel located at row 72, column 78. The example illustrates the effect of perched saturation. The pixel only generates runoff when it is in the surface-saturated state (between times 1006:00 and 1017:00), which can be identified because the top front is at the surface or because the degree of saturation is 1. Comparative analysis of different variables helps the user understand model evolution and identify possible errors.



The option to display pixel state is used in this example to explain the behavior of the pixel (72,78) presented in the previous image. Pixel state has been generated for six different times. The pixel is unsaturated in the first one (upper left, 1003:00). It reaches the perched-saturated state at time 1005:00 (upper center), and at time 1006:00 (upper right) the top front reaches the surface and the pixel becomes surface-saturated. Penetration of the wetting front can be observed in the fourth window (lower left, 1014:00). The image at 1020:00 shows how the pixel becomes perched-saturated again when rainfall stops. Eventually (lower right, 1030:00), the wetting front reaches the water table level.

APPENDIX 5

Software Documentation

This appendix presents the software documentation of the RIBS package. The intended readers are future programmers who work in system maintenace or expansion, or in higher-level applications which use the modeling capabilities of RIBS. As it is customary in object-oriented programming, the documentation is centered around the objects that integrate the system.

Three aspects are presented for every object: purpose, object structure and object methods. The *purpose* section briefly summarizes the intended use of the object. The *object structure* section lists object variables, their meaning and units. The *object methods* section describes the member functions of the object. For every function, the following information is provided:

- Calling syntax.
- Argument list: types, meaning and units.
- Preconditions: variables which are assumed to be computed when the function is called.
- Objective: brief description of the purpose of the method.
- Return value: meaning and units.
- Side effect: any changes originated by the function in the object or in related objects.

- Algorithm: detailed pseudo-code description of the algorithm chosen for implementation.

Since several objects are only intended to store information and do not have behavior associated to them, the methods section is optional.

A5.1 Objects of the distributed simulator

The relationships between the objects of the distributed simulator are represented in Figure A5.1. Details for every object are presented hereof.

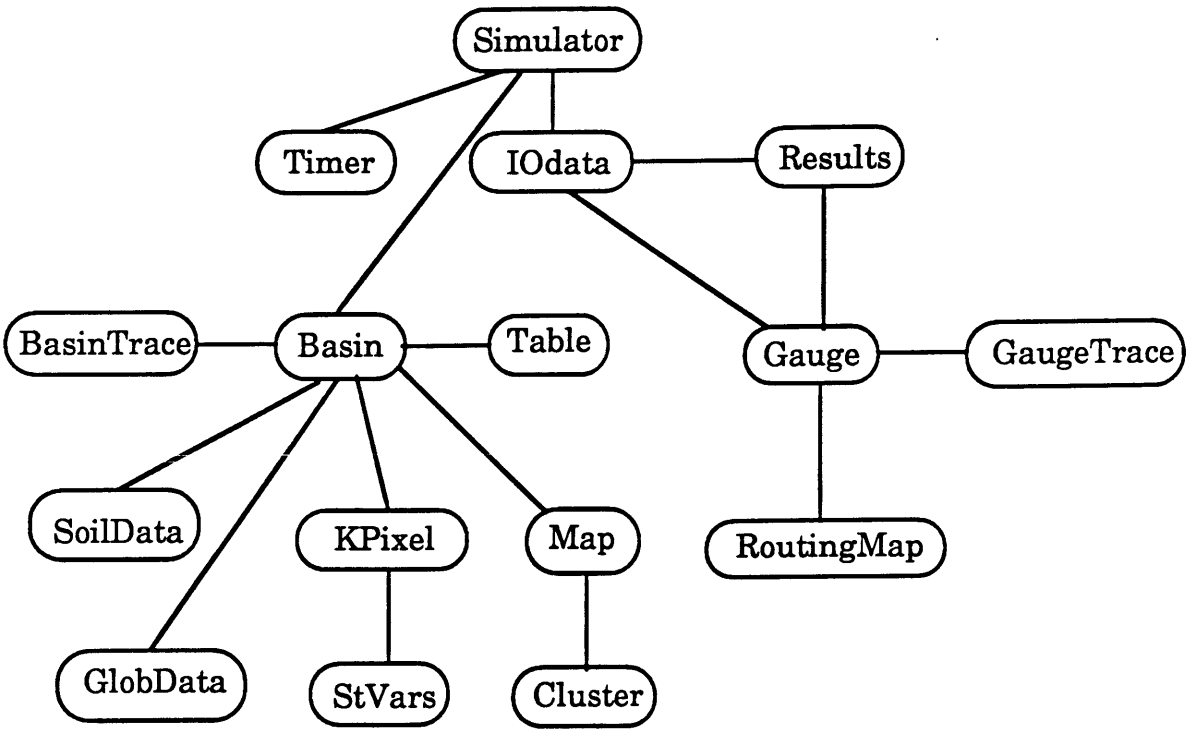


Figure A5.1 *Hierarchical relationships between the objects of the Distributed Basin Simulator*

GlobData Object

Purpose

Stores the variables which are uniform throughout the basin

Object structure

double dx;	Pixel x size in m
double dy;	Pixel y size in m
double pixarea;	Pixel area in m^2
double f;	Rate of decrease of permeability with depth in mm^{-1}
double anis_ratio;	Anisotropy ratio

StVars Object

Purpose

Stores the pixel state variables

Object structure

double Nf;	Depth of the wetting front in mm
double Nt;	Depth of the top front in mm
double d_Mt;	Moisture content of the pixel in mm

KPixel Object

Purpose

The *KPixel* represents a soil column whose behavior is governed by the kinematic model of infiltration

Object structure

General data

GlobData *global;	A pointer to global basin data
StVars *stv;	Pointer to state variables of the pixel
double dt_calc;	Computation time step
int state;	Indicator of the state of the pixel

Soil properties

double k0n;	Hydraulic conductivity normal to the surface in mm/h
double e;	Pore size distribution index (epsilon) in Brooks-Corey
double ths;	Saturation moisture content
double thr;	Residual moisture content in Brooks-Corey

Pixel properties

double alpha;	Slope angle
double s;	Tangent of slope angle
double cos_alpha;	Cosine of slope angle
double sin_alpha;	Sine of slope angle
double str_dist;	Distance to nearest stream in m
double nwt;	Water table depth for every pixel in mm

Values deduced from geometry and state variables

double W;	Width of the pixel normal to the flow in m
double L;	Length of the pixel parallel to the flow in m
double i_init_recharge;	Initial recharge rate in mm/h
double timeg;	Time of travel in hours
double Nstar;	Saturation level in mm
double N_Ristar;	Saturation level in mm for initial recharge rate
double d_Mu;	Unsaturated moisture content of the pixel in mm

doubleqn_Nf; Normal flow at the wetting front level in mm/h

State variable increments

double incr_Nf; Rate of change of the depth of the wetting front in mm/s

double incr_Nt; Rate of change of the depth of the top front in mm/s

double incr_d_Mt; Rate of change of moisture content of the pixel in mm/s

double incr_M_rain; Change of Mt due to rain in mm/s

double incr_M_Q; Change of Mt due to balance of i_Qin and i_Qout in mm/s

double incr_M_front; Change of Mt due to front advance in mm/s

Mass balance variables:

i_ : intensity variable mm/h

d_ : depth variable mm

v_ : volume variable m^3

double i_rain; Rainfall rate at every time step in mm/h

double d_rain; Pixel rainfall in mm

double i_eq_rain; Equivalent rainfall rate at every time step in mm/h

double i_infiltration; Hillslope infiltration (intensity) in mm/h

double d_infiltration; Depth of infiltration in the pixel in mm

double d_rain_runoff; Hortonian runoff in the pixel in mm

double d_M_runoff; Subsurface runoff in the pixel in mm

double d_runoff; Runoff generated in the pixel in mm

double v_runoff; Pixel contribution to surface runoff in m3

double i_runoff; Average rate of surface runoff in mm/h

double i_Qin; Discharge into the pixel in mm/h

double i_Qout; Moisture out of the pixel (discharge) in mm/h

Object methods

Function: *hillpix(pxl,numpix)*

Arguments: pointer to KPixel data structure.

int: Number of pixel identification.

Preconditions: Nf,Nt,d_Mt,nwt,soil parameters

Objective: Compute pixel evolution for a time step and evaluate pixel runoff.

Return value: double: Runoff volume during the time step

Side effects: Front position and moisture content are updated. Other related values, such as Nstar, Re, etc. are also updated according to new pixel state.

Algorithm:

```
obtain rainfall depth
if water table at the surface
  runoff = rainfall depth
else
  obtain equivalent rain
  compute subsurface outflow
  compute flow at wetting front
  update front position
  update moisture evolution
  evaluate runoff
update trace
```

Function: *strpix(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: soil parameters

Objective: Compute pixel evolution for a time step and evaluate pixel runoff in a stream pixel

Return value: double: Runoff volume during the time step

Side effects: none

Algorithm:

```
obtain rainfall depth
runoff equals rainfall depth
```

Function: *Re(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: d_Mu,Nf,Nt,i_init_recharge,i_rain,soil parameters

Objective: obtain the steady rainfall rate which would produce the same moisture status

Return value: double: Equivalent recharge rate (mm/h)

Side effects: none

Algorithm:

```
if top front is close to the surface
  Re = i_rain
else
 apply equation:
  if base of power function is positive
    apply formula
  else
    Re = Ri
```

Function: *Theta(pxl,r,n)*

Arguments: pointer to KPixel data structure.
double: Rainfall rate in mm/h
double: depth in mm

Preconditions: soil parameters

Objective: obtain moisture content at some depth for a given a steady rainfall rate

Return value: double: Moisture content (dimensionless)

Side effects: none

Algorithm:
apply Theta equation

Function: *Mu(pxl,r,n)*

Arguments: pointer to KPixel data structure.
double: rainfall rate in mm/h
double: depth in mm

Preconditions: soil parameters

Objective: obtain pixel moisture content in the unsaturated area up to some depth for a given equivalent rainfall rate

Return value: double: moisture depth in mm

Side effects: none

Algorithm:
apply Mu equation

Function: *sub_outflow(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: Nf,Nt,i_eq_rain,soil parameters

Objective: obtain subsurface outflow in homogeneous terrain

Return value: double: Subsurface outflow in mm/h

Side effects: The variables i_Qout_sat and i_Qout_unsat are assigned their values

Algorithm:
if there is unsaturated area
compute i_Qout_unsat
if there is saturated area
compute i_Qout_sat

Function *flow_at_front(pxl)*

Arguments: KPixel data structure.

Preconditions: Nf, Nt, d_Mt, nwt, i_eq_rain, soil parameters

Objective: compute normal flow at the wetting front level

Return value: double: normal flow in mm/h

Side effects: none

Algorithm:
if the wetting front is at nwt or N_Ristar
normal flow is zero.
otherwise
if there is no saturated zone
normal flow is equivalent rain
if the saturated zone is very thin
normal flow is normal hydraulic conductivity at the front.
else
normal flow is average flow over the saturated area.

Function: *front_transition(pxl,numpix)*

Arguments: pointer to KPixel data structure.
int: Number of pixel identification.

Preconditions: Nf, Nt, d_Mt, nwt, i_eq_rain, soil parameters

Objective: obtain rates of advance of wetting and top front according to the pixel initial state (proposed increments)

Return value: void

Side effects: evaluate incr_Nf, incr_Nt

Algorithm:
if Nf is at the water table
wetting front advance is zero
flow in saturated area is zero
apply deep-saturated equation to obtain top front advance:
if denominator is small (close to N*)
if pixel is filling
iterate with Nt increments until Nt>N*

```
        if pixel is draining
          top front increment is zero (there is no instability)
  elseif Nf is at the saturation level for Ri (NRi*)
    wetting front advance is zero
    flow in saturated area is zero
    apply deep-saturated equation to obtain top front advance:
      if denominator is small (close to N*)
        if pixel is filling
          iterate with Nt increments until Nt>N*
        if pixel is draining
          top front increment is zero (there is no instability)
  else
   if the pixel does not have a saturated zone
     if flow at front is smaller than equivalent rain (still unsaturated)
      apply unsaturated equation to obtain wetting front advance:
        if denominator is smaller than zero (pixel has drained out)
          wetting front advance is zero
        else
          wetting front advance is given by equation
      top front advance is equal to wetting front advance
     else (saturation begins at this time step)
      apply unsaturated equation to obtain wetting front advance:
        if denominator is very small (seems absurd, but it happens!)
          wetting front advance is zero
        else
          wetting front advance is given by equation
      top front advance is given by limit equation
   else (the pixel has a saturated zone)
      apply saturated equation to obtain wetting front advance:
        if denominator is very small (Nf very close to NRi*)
          set wetting front advance to reach NRi* (maximum)
        else
          wetting front advance is given by equation
      apply saturated equation to obtain top front advance:
        if denominator is very small (Nt very close to N*)
          if flow in saturated zone is greater than equivalent rain
           (front goes down)
            top front advance is given by limit equation
          else (front goes up)
            set top front advance to reach N* (maximum)
        else
          top front advance is given by equation
```

Function: *moisture_transition(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: Nf, Nt, d_Mt, incr_Nf, incr_Nt, i_eq_rain, soil parameters, i_Qin, i_Qout

Objective: obtain terms in the moisture balance equation: moisture increments due to front advance, infiltration and subsurface flows.

Function should be evaluated with updated front positions

Return value: void

Side effects: the variables incr_M_front, incr_M_rain, incr_M_Q and incr_d_Mt are assigned their values

Algorithm:

incr_M due to front advance is incr_Nf times average moisture
obtain infiltration capacity
incr_M due to infiltration is min(rain, infiltration capacity)
incr_M due to subsurface flow in inflow minus outflow

Function: *inf_capacity(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: Nf, Nt, d_Mt, nwt, i_eq_rain, qn_Nf, soil parameters

Objective: obtain maximum surface infiltration capacity for the pixel

Return value: double: infiltration capacity in mm/h

Side effects: none

Algorithm:

if top front is at the surface
  infiltration capacity is flow in the saturated area / cos(alpha)
else
  infiltration capacity is normal hydraulic conductivity / cos(alpha)

Function: *comp_runoff(pxl,numpix)*

Arguments: pointer to KPixel data structure.
int: Number of pixel identification.

Preconditions: Nf, Nt, d_Mt, incr_Nf, incr_Nt, incr_d_Mt, nwt, i_eq_rain, soil parameters

Objective: obtain both modes of runoff (surface runoff and return flow) and verify that all other variables are mutually consistent. Fix variables if inconsistencies are found.

Return value: double: runoff depth during the time step in mm

Side effects: the variables d_rain_runoff and d_M_runoff are assigned their values. Corrects the proposed Nf, Nt and d_Mt values according to the results of the moisture balance

Algorithm:

obtain return flow:

```
obtain moisture content of the fully saturated pixel: max_sat
obtain moisture content due to the initial state d_Mu_min
obtain moisture error: err_mois = (max_sat-d_Mu_min) - d_Mt
if moisture error is negative
  if Nf is very small (pixel was dry: all input is subsurface moisture)
    increase iteratively Nf to accomodate subsurface inflow:
    if the pixel becomes fully saturated (yes, it happens!)
      wetting front is at the water table
      top front is at the surface
      total moisture is max_sat
      the remaining is return flow
  else (subsurface inflow causes return flow)
    top front is at the surface
    error moisture is return flow
    total moisture is max_sat
else (moisture error is positive)
  return flow is zero
  if moisture error is very small (saturation, for practical purposes)
    top front is at the surface
    total moisture is max_sat
  else (moisture content corresponds to unsaturated, verify Nt and Re)
    if top front is at the surface (verify Nt)
      (front dynamics gave saturation, but subsurface outflows
      emptied the pixel)
      obtain iteratively a new top front position, with Nt<N*
    else (verify Re)
      obtain unsaturated moisture
      obtain equivalent rainfall
      if Re is smaller than Ri
        (unsaturated area dried out because of subsurface outflow)
        obtain iteratively a new top front position, with Re>Ri
        if there was a saturated area
          if Nt becomes > Nf and Re is still < Ri
            pixel has completely dried out
            reset front positions and moisture content to zero
          else (no saturated area)
            pixel has completely dried out
            reset front positions and moisture content to zero
      elseif equivalent rain is smaller than normal conductivity
        obtain N*
        if N*<Nt
          obtain iteratively a new top front position, with Nt<N*
      else (Nt is very small -> saturation, for practical purposes)
        top front is at the surface
        define wetting front according to moisture content
obtain surface runoff
```

Function: *outflow_complement(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: Nf,Nt,nwt,soil parameters

Objective: evaluate subsurface outflow from the pixel due to pressure distribution in the saturated zone. Only this pixel is considered

Return value: double: subsurface outflow in mm/h

Side effects: none

Algorithm:

```
if there is saturated area
  evaluate flow according to equation
else
  flow is zero
```

Function: *check_moisture(pxl,numpix)*

Arguments: pointer to KPixel data structure.

int: Number of pixel identification.

Preconditions: Nf, Nt, d_Mt, nwt, i_Qout, soil parameters

Objective: solve the problem of the empty pixel: moisture transition is such that the pixel dries out.

Return value: void

Side effects: corrects i_Qout, and Nf, Nt, Mt are reset to zero

Algorithm:

```
if i_Qout is larger than incr_Mt in absolute value
  i_Qout is set to -d_Mt
  wetting and top fronts are reset to the surface.
  moisture content is reset to zero
```

Function: *get_Nf(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel state

Objective: obtain wetting front postion

Return value: double: wetting front depth in mm

Side effects: none

Algorithm:

```
return wetting front depth
```

Function: *get_Nt(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel state

Objective: obtain top front postion

Return value: double: top front depth in mm

Side effects: none

Algorithm:
  return top front depth

Function: *get_Mt(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel state

Objective: obtain moisture content in the pixel

Return value: double: moisture content in mm

Side effects: none

Algorithm:
  return moisture content

Function: *get_runoff(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: runoff generation

Objective: obtain runoff generation

Return value: double: runoff generatiuon rate in mm/h

Side effects: none

Algorithm:
  return runoff generation rate

Function: *get_rain(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: rainfall rate

Objective: obtain rainfall rate

Return value: double: rainfall rate in mm/h

Side effects: none

Algorithm:
  return rainfall rate

Function: *get_k0(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties

Objective: obtain surface normal hydraulic conductivity

Return value: double: hydraulic conductivity in mm/h

Side effects: none

Algorithm:
  return hydraulic conductivity

Function: *get_ic(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain pixel maximum infiltration capacity

Return value: double: infiltration capacity in mm/h

Side effects: none

Algorithm:
  return infiltration capacity

Function: *get_ff(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain normal flow at the level of the wetting front

Return value: double: flow rate in mm/h

Side effects: none

Algorithm:
  return flow at wetting front

Function: *get_upper_deficit(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain moisture deficit above the wetting front

Return value: double: moisture deficit in mm

Side effects: none

Algorithm:
  if water table at the surface
    return zero
  if wetting front at the surface
    return zero
  get maximim saturated moisture content
  get initial moisture content
  return moisture deficit

Function: *get_upper_saturation(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain moisture deficit above the wetting front % with respect to saturation

Return value: double: saturation (dimensionless)

Side effects: none

Algorithm:

```
if water table at the surface
  return one
if wetting front at the surface
  return zero
get maximim saturated moisture content
get initial moisture content
return ratio of moisture content to maximum capacity
```

Function: *get_lower_deficit(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain moisture deficit between the wetting front and the water table

Return value: double: moisture deficit in mm

Side effects: none

Algorithm:

```
if water table at the surface
  return zero
if wetting front at the surface
  return zero
get maximim saturated moisture content up to the wetting front
get maximim saturated moisture content up to the water table
get initial moisture content up to the wetting front
get initial moisture content up to the water table
return lower moisture deficit
```

Function: *get_total_deficit(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain total moisture deficit in the column

Return value: double: moisture deficit in mm

Side effects: none

Algorithm:
  if water table at the surface
    return zero
  if wetting front at the surface
    return zero
  get maximim saturated moisture content up to the water table
  get initial moisture content up to the water table
  return moisture deficit

Function: get_total_saturation(pxl)

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties, pixel state

Objective: obtain moisture deficit in the soil column (% with respect to saturation)

Return value: double: saturation (dimensionless)

Side effects: none

Algorithm:
  if water table at the surface
    return one
  if wetting front at the surface
    return zero
  get maximim saturated moisture content up to the water table
  get initial moisture content up to the water table
  return ratio of moisture content to maximum capacity

Function: *get_stream_dist(pxl)*

Arguments: pointer to KPixel data structure.

Preconditions: pixel properties

Objective: obtain distance to nearest stream

Return value: double: distance to nearest stream in m

Side effects: none

Algorithm:
  return stream distance

SoilData Object

Purpose

Stores soil properties arranged in soil groups

Object structure

int maxclass;	Number of soil classes
double *k0m;	Array [maxclass]: Hydraulic conductivity
double *thsm;	Array [maxclass]: Saturation moisture content
double *thrm;	Array [maxclass]: Residual moisture content
double *em;	Array[maxclass]: Brooks-Corey parameter

Table Object

Purpose

Stores the order of computation which should be followed in the basin state loop. Keeps track of row and column of every pixel and of the position in the basin arrays

Object structure

int h_pixels;	Number of hillslope pixels
int s_pixels;	Number of stream pixels
int *list[2];	Array[2] : pointers to order of computation [0]: column , [1]: row
int *self;	Array [n_pixels + s_pixels]: position of pixel in basin array
int *next;	Array [n_pixels + s_pixels]: position of downstream pixel in basin array

Cluster Object

Purpose

Represents the structure of a cluster in the raster format.

Object structure

int row;	Row of the cluster
int col;	Column of first position in the cluster
int posval;	Position in basin arrays of first cell in the cluster
int ncells;	Number of cells

Map Object

Purpose

Stores the equivalence between the raster format and the basin array.

Provides functions to transform one into the other

Object structure

int nrows;	Number of rows in the basin
int ncols;	Number of columns in the basin
int nclus;	Number of clusters in the raster
int *rows;	Array [nrows]: First cluster of every row
Cluster *clus;	Array [nclus]: Clusters

Object methods

Function: *gen_map(map,hr)*
Arguments: pointer to Map data structure.

pointer to Raster data structure.

Preconditions: none

Objective: allocate memory and initialize the basin map given the raster structure of the basin. The row array contains the first cluster of every row, and the clusters array contains row, column and position of first element and number of cells in the cluster

Return value: void

Side effects: pointers are allocated, and basin map is initialized

Algorithm:

```
get number of rows, columns and clusters
allocate memory for rows and cluster arrays
for all clusters in the raster:
 if it is the first cluster in the row
   initialize row value
 initialize cluster data structure
```

Function: *get_pos(map,ir,ic)*

Arguments: pointer to Map data structure.

int: row index

int: column index

Preconditions: Map data structure. Row and column are assumed greater than zero

Objective: return the index of the basin arrays corresponding to a pixel identified by its row and column

Return value: int: array index, or negative if pixel is outside the basin

Side effects: none

Algorithm:

```
if row is greater than number of rows, return -1
for all clusters in that row:
  if column is between the first and last positions of the cluster
    return position
return -1 (pixel is not in the clusters of that row)
```

BasinTrace Object

Purpose

BasinTrace stores information about how runoff is generated in the basin. It keeps track of number of pixels in every state and runoff volume generated in every mode.

Object structure

General data

int npixels;	Number of pixels in the basin
double dt_calc;	Computation time step
double fac_res;	Factor in the result = pixel area/secs in result time step
FILE *filnum;	Pointer to file for number of pixels
FILE *filvol;	Pointer to file for runoff volume

Global values

double ahr;	Cummulative (in time) pixel rainfall (spatial) average
double ahrf;	Cummulative (t) pixel runoff (sp) average
double cumrr;	Cumulative rainfall in all the basin every time step
double trvol;	Total runoff generated in every time step

Water table at the surface

int nwts;	Number of pixels with water table at the surface
double d_wts1;	Volume of infiltration-excess runoff generated
int nwts2;	Number of pixels with water table at the surface generating return flow
double d_wts2;	Volume of return flow generated

Stream pixels

int nstp;	Number of stream pixels
double d_stp1;	Volume of infiltration-excess runoff generated
int nstp2;	Number of stream pixels generating return flow
double d_stp2;	Volume of return flow generated

Unsaturated pixels

int nust;	Number of unsaturated pixels
int nust1;	Number of unsaturated pixels generating infiltration-excess runoff
double d_ust1;	Volume of infiltration-excess runoff generated

Perched-saturated pixels

int ndst;	Number of perched-saturated pixels
int ndst1;	Number of perched-saturated pixels generating infiltration-excess runoff
double d_dst1;	Volume of infiltration-excess runoff generated

Surface-saturated pixels

int nsst;	Number of surface-saturated pixels
int nsst1;	Number of surface-saturated pixels generating infiltration runoff
double d_sst1;	Volume of infiltration-excess runoff generated
int nsst2;	Number of surface-saturated pixels generating return flow
double d_sst2;	Volume of return flow generated

Fully-saturated pixels

int nwst;	Number of fully-saturated pixels
double d_wst1;	Volume of infiltration-excess runoff generated
int nwst2;	Number of fully-saturated pixels generating return flow
double d_wst2;	Volume of return flow generated

Object methods

Function: *btr_update(tr,pixel)*
Arguments: pointer to BasTrace data structure.
pointer to KPixel data structure.
Preconditions: pixel state variables and runoff
Objective: update the basin trace variables with the results for that pixel
Return value: void
Side effects: trace variables (number of pixels and runoff volume) updated
Algorithm:
according to pixel state

increment counter of number of pixels in the state
increment surface runoff for the state
increment subsurface runoff for the state
increment global counters

Function: *btr_res_dump(tr)*

Arguments: pointer to BasTrace data structure.

Preconditions: all basin trace information

Objective: writing basin trace results to the corresponding files

Return value: void

Side effects: none

Algorithm:
print number of pixels in every state
print runoff volume generated by pixels in every state

Function: *btr_reset(tr)*

Arguments: pointer to BasTrace data structure.

Preconditions: none

Objective: reset basin trace after a time step

Return value: void

Side effects: all trace variables set to zero

Algorithm:
set everything to zero

Timer Object

Purpose

The Timer keeps track of the basic time settings for the simulation, and provides functions to transform from actual time in hours to simulation or results time steps.

Object structure

double start_hour; Generic time start in hours

double final_hour;	Generic last time in hours
double dt_calc;	Time step of calculations in min(read)/hour(used)
double dt_res;	Time step of results in min(read)/hour(used)
double dt_res_sec;	Time step of results in sec
double t0;	Time at the beginning of every step

Object methods

Function: *sim_step(timer,t)*
Arguments: pointer to Timer data structure.
double: current time in h
Preconditions: timer data
Objective: obtain the simulation step corresponding to a given time
Return value: int: number of simulation step
Side effects: none
Algorithm:
obtain simulation step

Function: *res_step(timer,t)*
Arguments: pointer to Timer data structure.
double: result time in h
Preconditions: timer data
Objective: obtain the result step corresponding to a given time
Return value: int: number of result step
Side effects: none
Algorithm:
obtain result step

Function: *res_step_hyd(timer,t)*
Arguments: pointer to Timer data structure.
double: result time in h
Preconditions: timer data
Objective: obtain the result step for the extra gauge corresponding to a given time
Return value: int: number of result step

Side effects: none

Algorithm:
obtain result step

Basin Object

Purpose

Stores the distributed variables in the basin. It controls interactions between pixels during model inference. The *Basin* object also manages several children objects, such as *BasinTrace*, *Map* and *Pixel*.

Object structure

Related objects

GlobData *global;	Pointer to global data
SoilData *soil;	Pointer to soil types
KPixel *pixel;	Pointer to pixel
Table order;	Table with the order of computations
RasHead hdr;	Raster head to store distributed variables
Map header_map;	Basin map
Timer *timer;	Pointer to simulation timer
BasinTrace *trace;	Pointer to basin trace data structure

Basin variables and parameters

int npixels;	Total number of pixels in the basin
int nraster;	Total number of pixels in the raster
double width[8];	Array: Pixel width in m in every direction
double length[8];	Array: Pixel length in m in every direction
double hill_vel;	Hillslope velocity (km h^{-1})
double stre_vel;	Stream velocity (km h^{-1})
double vel_ratio;	Ratio of stream to hillslope velocity
double vel_coef;	Coefficient of the vel-disch relationship
double vel_exp;	Exponent of the vel-disch relationship

double baseflow; Baseflow discharge in $m^3 s^{-1}$(artificially added)
double recharge; Initial recharge rate

Derived basin data

double timegmax; Maximum travel time : time of response
double dist_hill_max; Maximum distance for the fartest pixel
double dist_stream_max; Value for the fartest pixel

Basin arrays: distributed variables and parameters

int *dir; Array [nraster]: Pointer
int *class; Array [nraster]: Soil type
double *slope; Array [nraster]: Slope
double *nwtm; Array [nraster]: Water table depth in mm
double *nf; Array [nraster]: Wetting front depth in mm
double *nt; Array [nraster]: Top front in mm
double *d_Mtm; Array [nraster]: Moisture content in the pixel in mm
double *v_runoff; Array [nraster]: Total runoff volume in every pixel in m^3
double *v_retflow; Array [nraster]: Return flow volume in every pixel in m^3
double *rf; Array [nraster]: Average runoff generated at every pixel in mm/h
double *i_Qinm; Array [nraster]: Subsurface discharge into the pixel in mm/h
double *i_Qoutm; Array [nraster]: Parallel discharge out of the pixel in mm/h
double *distg; Array [nraster]: Distance to gage in m
double *dists; Array [nraster]: Distance to stream in m
double *rain; Array [nraster]: Rainfall intensity in mm/h

Object methods

Function: *init_stv_pixel(basin,ip)*
Arguments: pointer to Basin data structure.
int: Pixel array index
Preconditions: basin state arrays

Objective: initialize pixel state variables

Return value: void

Side effects: the stv pointer in the pixel data structure is initialized

Algorithm:
Nf, Nt, Mt values are taken from the basin arrays

Function: *init_pixel(basin,ip)*

Arguments: pointer to Basin data structure.

Preconditions: State arrays, properties arrays

Objective: make the pixel point to basin position ip

Return value: void

Side effects: the pixel data structure is fully initialized

Algorithm:
initialize state variables
get soil class
initialize soil properties
obtain initial recharge rate
obtain unsaturated moisture content
obtain NRi*

Function: *hill_loop(basin)*

Arguments: pointer to Basin data structure.

Preconditions: Map structure, basin state arrays (previous time step), properties arrays, rainfall array

Objective: evaluate state evolution and runoff generation during a computation time step for al pixels in the basin

Return value: void

Side effects: state arrays and runoff generation array are updated

Algorithm:
for all pixels in basin order:
obtain array position
obtain row and column
get rainfall rate from array
initialize pixel
evaluate pixel runoff
if basin trace is active
update trace
store results in arrays
store i_Qout as subsurface inflow for downslope pixel
reset subsurface inflow to zero

Function: *stre_loop(basin)*

Arguments: pointer to Basin data structure.

Preconditions: Map structure, basin state arrays (previous time step), rainfall array

Objective: evaluate state evolution and runoff generation during a computation time step for stream pixels in the basin

Return value: void

Side effects: runoff generation array is updated

Algorithm:

```
for all pixels in basin order:
   obtain array position
   obtain row and column
   get rainfall rate from array
   get subsurface inflow from array
   evaluate pixel runoff
   if basin trace is active
     update trace
   store runoff in array
   reset subsurface inflow to zero
```

Function: *bas_start(basin)*

Arguments: pointer to Basin data structure.

Preconditions: number of elements in arrays

Objective: allocate memory for the arrays and initialize them

Return value: void

Side effects: array pointers initialized

Algorithm:

```
call calloc for evary array, checking for errors
```

Function: *para_flow_loop(basin)*

Arguments: pointer to Basin data structure.

Preconditions: Map structure, basin state arrays, properties arrays

Objective: obtain subsurface transfer of moisture between pixels due to lateral pressure gradients in the saturated zone

Return value: void

Side effects: the subsurface inflow array is initialized for all pixels containing the corresponding flow

Algorithm:

```
for all pixels in basin order:
```

```
obtain array position
initialize state variables in pixel
obtain outflow
obtain array position of the downslope pixel
initialize state variables in pixel
 get subsurface inflow from array for downslope pixel
obtain outflow for downslope pixel
evaluate pixel runoff
obtain average lateral hydraulic conductivity
obtain total outflow as outflow from the pixel minus
        outflow from the downslope pixel
store subsurface inflow in downslope pixel
```

Function: *get_area(basin,ip,i,j)*

Arguments: pointer to Basin data structure.

int: Pixel array index

int: Pixel row

int: Pixel column

Preconditions: Map structure

Objective: obtain contributing area to a pixel in the basin

Return value: int: number of contributing pixels or zero, if not in basin

Side effects: none

Algoritm:

```
 if pixel is outside basin return 0
 else
  initialize area to 1 (that pixel)
  get pointer from the array
  for all surrounding pixels:
     obtain row and column
     get pixel array index
     if pixel is in basin
        if surrounding pixel drains to pixel
           increment area by the area of surrounding pixel
  return area
```

Function: *init_bas_trace(basin)*

Arguments: pointer to Basin data structure.

Preconditions: number of pixels, simulation timer

Objective: allocate memory and initialize the basin trace data structure

Return value: void

Side effects: files open, pointers initialized

Algoritm:

Open files
Allocate memory
Set initial values

Function: *end_bas_trace(basin)*

Arguments: pointer to Basin data structure.

Preconditions: files open

Objective: terminate the basin trace data structure

Return value: void

Side effects: files closed, pointers released

Algoritm:
close files
free memory

RoutingMap Object

Purpose

Stores the pixels located upstream of the gauge, sorted according to the inverse of the recursive 'drains to' relationship.

Object structure

int n_pixels;	Number of pixels
int *pxl;	Array [n_pixels]: Position in basin array of pixel
double *distg;	Array [n_pixels]: Distance from pixel to gauge along channels

Results Object

Purpose

Stores hyetographs and hydrographs obtained for a gauge. It keeps track of previous and current rainfall and previous, current and future hydrographs.

Object structure

General data

int limit;	Size of results array
int iimax;	Last non-zero position of results array

Hydrograph and hyetograph arrays

double *phydro;	Array [limit]: Previous hydrograph (expected response) in m^3/s
double *hhydro;	Array [limit]: Response with measured rain in m^3/s
double *shydro;	Array [limit]: Response with measured and forecasted rain in m^3/s
double *prr;	Array [limit]: Previous rainfall in mm
double *crr;	Array [limit]: Rainfall in the time step in mm

GaugeTrace Object

Purpose

Gauge trace stores the decomposition of the hydrograph into the different modes of runoff generation

Object structure

General data

int npixels;	Number of pixels in the gauge
double dt_calc;	Computation time step
double fac_res;	Factor in the result = Pixel area/secs in result time step
FILE *monitor30;	Pointer to file

Hydrograph arrays

double *ret_perm;	Array [limres]: Return flow in permanently saturated pixels in m^3/s
double *ret_temp;	Array [limres]: Return flow in temporary saturated pixels in m^3/s
double *iex_perm;	Array [limres]: Infiltration-excess runoff in permanently saturated pixels in m^3/s
double *iex_temp;	Array [limres]: Infiltration-excess runoff in temporary saturated pixels in m^3/s

Object methods

Function: *gtr_update(tr,bas,ras_pos,ih)*
Arguments: pointer to GaugeTrace data structure.
pointer to Basin data structure.
int: position in basin arrays
int: time index
Preconditions: basin state and behavior, result time step
Objective: update the gauge trace data structure for a pixel
Return value: void
Side effects: the trace results are updated
Algorithm:

```
if saturation is permanent
   update permanent surface runoff
   update permanent return flow
else
   update transient surface runoff
   update transient return flow
```

Gauge Object

Purpose

Obtains hydrographs at a point in the basin. It performs the distributed convolution operation on the pixels upstream the gauge.

Object structure

char identification[80];	Denomination of the gauge
Results *res;	Pointer to hydrographs
Basin *basin;	Pointer to the basin
GaugeTrace *trace;	Pointer to gauge trace data structure
RoutingMap order;	Pointer to routing map
double timegmax;	Maximum time of travel
double dist_stream_max;	Maximum distance to stream

Object methods

Function: *init_results(gauge)*
Arguments: pointer to Gauge data structure.
Preconditions: basin velocities and timer data
Objective: allocate memory and initialize the Results data structure
Return value: void
Side effects: maximum number of points and pointers initialized
Algorithm:
set velocity for Q=0
get result step for end of simulation plus time of response
allocate memory for the hydrographs
obtain gauge baseflow
initialize hydrographs to zero

Function: *free_results(gauge)*
Arguments: pointer to Gauge data structure.
Preconditions: Results data structure initialized
Objective: terminate the Results data structure and free memory

Return value: void

Side effects: all pointers set to NULL

Algorithm:
free all pointers

Function: *route_loop(gauge)*

Arguments: pointer to Gauge data structure.

Preconditions: basin velocities, basin runoff generation, rainfall, timer data

Objective: evaluate the distributed convolution for a computation time step

Return value: void

Side effects: none

Algorithm:
```
set common factor for runoff
for all pixels in gauge order:
  get basin array index
  obtain travel time
  obtain result step for rainfall
  obtain result step for discharge
  store rainfall result
  store discharge result
  if gauge trace is active
    update gauge trace
```

Function: *init_routing(gauge,i,j)*

Arguments: pointer to Gauge data structure.
int: row position
int: column position

Preconditions: basin map, timer data

Objective: initialize the Gauge data structure and children objects

Return value: void

Side effects: memory allocated, routing map built, and number of pixels and maximum distance to gauge initialized

Algorithm:
```
obtain basin array index for pixel (i,j)
obtain contributing area
allocate memory for order and distance to gauge
build the routing map
initialize maximum distance to gauge
for all pixels in routing map:
  obtain basin array index
```

verify distance to gauge
update maximum distance to gauge

Function: *free_routing(gauge)*

Arguments: pointer to Gauge data structure.

Preconditions: Gauge children initialized

Objective: free memory in the Gauge data structure

Return value: void

Side effects: none

Algorithm:
free pointers

Function: *build_route_map(gauge,ip,i,j,count,dist)*

Arguments: pointer to Gauge data structure.
int: basin array index
int: row position
int: column position
pointer to int: gauge index counter
double: current distance to gauge

Preconditions: basin arrays, basin map, contributing area

Objective: initialize the routing map. The routing map contains the pixels upstream the gauge sorted in the order in which they should be processed, with upstream pixels frist. The order is given by applying recursively the relation drains to.

Return value: void

Side effects: order and distance to gauge data structures are initialized for the gauge

Algorithm:
get basin pointer
get length of the pixel
decrement counter
increment distance to gauge by pixel length
store current pixel in order array
store current distance in distance array
for all surrounding pixels:
get row and column
get basin array position
if surrounding pixel drains to current pixel
call build route map

Function: *init_gau_trace(gauge)*

Arguments: pointer to Gauge data structure.

Preconditions: Results data structure initialized, timer data

Objective: allocate memory and initialize the gauge trace data structure

Return value: void

Side effects: pointers and variables in gauge trace initialized

Algorithm:
  allocate pointers
  initialize variables

Function: *end_gau_trace(gauge)*

Arguments: pointer to Gauge data structure.

Preconditions: Gauge trace updated

Objective: write results and terminate the gauge trace data structure

Return value: void

Side effects: memory released and pointers set to NULL

Algorithm:
  compose file name
  open file
  for all points in trace results array:
    write values
  close file
  free pointers

Function: *set_velocity(gauge,ihour)*

Arguments: pointer to Gauge data structure.
  int: result step

Preconditions: basin velocities, results arrays up to ihour, timer data

Objective: set basin hillslope and stream velocities according to the power relation. The equation takes streamflow in gauge at time ihour

Return value: void

Side effects: basin variables stre_vel and hill_vel set

Algorithm:
  obtain streamflow at ihour
  obtain exponent in power law
  apply equation to hillslope velocity
  obtain stream velocity from hillsope velocity

Function: *route_hydrograph(gauge,factorq,factorr)*

Arguments: pointer to Gauge data structure.
double: factor to apply to streamflow
double: factor to apply to rainfall

Preconditions: basin velocities, basin runoff generation, rainfall, timer data

Objective: evaluate the distributed convolution for a rainfall time step

Return value: void

Side effects: Results data structure updated

Algorithm:
for all pixels in gauge order:
get basin array index
obtain travel time
obtain result step for rainfall
obtain result step for discharge
store rainfall result
store discharge result

Function: *init_pixevol(gauge,i,j)*

Arguments: pointer to Gauge data structure.
int: row position
int: column position

Preconditions: basin map, timer data

Objective: Initialize the Gauge data structure and children objects for a basin composed of a single pixel

Return value: void

Side effects: memory allocated, routing map built, and number of pixels and maximum distance to gauge initialized

Algorithm:
obtain basin array index for pixel (i,j)
set contributing area to 1
allocate memory for order and distance to gauge
initialize the routing map
initialize maximum distance to gauge

Function: *route_pix(gauge,factorq,factorr)*

Arguments: pointer to Gauge data structure.
double: factor to apply to streamflow
double: factor to apply to rainfall

Preconditions: Pixel runoff generation, rainfall, timer data

Objective: Evaluate instantaneous hydrograph contribution during a rainfall time step

Return value: void

Side effects: Results data structure updated

Algorithm:

```
for all pixels in gauge order:
  get basin array index
  obtain result step
  store rainfall result
  store discharge result
```

Function: *set_distance(gauge)*

Arguments: pointer to Gauge data structure.

Preconditions: basin velocities, maximum distances

Objective: evaluate maximum distance to outlet for the contributing area

Return value: void

Side effects: variable timegmax updated

Algorithm:

```
apply equation
```

Function: *unit_hydrograph(gauge,factorq,factorr)*

Arguments: pointer to Gauge data structure.

double: factor to apply to streamflow

double: factor to apply to rainfall

Preconditions: basin velocities, basin state arrays, timer data

Objective: evaluate hortonian runoff generation and the distributed convolution for uniform rainfall of unit intensity during a rainfall time step

Return value: void

Side effects: results data structure updated

Algorithm:

```
for all pixels in gauge order:
  get basin array index
  initialize pixel data structure
  obtain travel time
  get local runoff generation for unit rainfall
  obtain result step for rainfall
  obtain result step for discharge
```

store rainfall result
store discharge result

IOdata Object

Purpose

The IOdata object maintains information about input and output operations: paths, file names, file pointers, etc.

Object structure

Input/Output Variables

char geomrpath[80];	Path for geomorphology variables
char stvarpath[80];	Path for basin state variables
char stvarpath_root[80];	Basic path for basin state variables
char mrainpath[80];	Path for measured rainfall
char frainpath[80];	Path for forecasted rainfall
char mraindir[80];	Measured rainfall directory
char fraindir[80];	Forecasted rainfall directory
char last_stvarpath[80];	Path for geomorphology variables
char mrainfile[80];	Measured rainfall file name
char frainfile[80];	Forecasted rainfall file name
char last_ntfile[80];	Wetting front file name
char last_nffile[80];	Top front file name
char last_d_Mtfile[80];	Moisture content file name
char last_rffile[80];	Runoff generation file name

Control variables

char last_meas_rain[8];	Time tag of last measured rain
char last_fore_rain[8];	Time tag of last forecasted rain rain
char last_state[8];	Time tag of last basin state
char fore_rain_label;	Forecasted rain Y or N
char first_time;	First computation loop Y or N
char ini_state;	Read initial state Y or N

char state[8];	Time tag of current time
char auto_list;	Automatic file listing Y or N
char inter_results;	Write intermediate results Y or N

Automatic listing of rainfall files

struct direct *(*mrainlist);	List of measured rain files
int num_mr_list;	Number of elements in the mrain list
int cur_mr_list;	Current element of the mrain list
struct direct *(*frainlist);	List of forecasted rain files
int num_fr_list;	Number of elements in the mrain list
int cur_fr_list;	Current element of the frain list

Gauge names

char **hydrpath;	Pointer to array of gauge paths
char **hydrofile;	Pointer to array of hydrograph file names
char **last_hydrofile;	Pointer to array of last hydrograph file names

File pointers for the user interface

FILE *nf_pipe;	Pipe to display wetting front
FILE *nt_pipe;	Pipe to display top front
FILE *mt_pipe;	Pipe to display moisture content
FILE *rf_pipe;	Pipe to display runoff generation
FILE **grafpipe;	Pointer to array of pipes to display hydrographs

Simulator Object

Purpose

Controls the simulation loops during model inference. It is also in charge of all input/output operations using the data stored in IOdata.

Object structure

Related objects

Basin *basin;	Pointer to basin data structure
IOdata *inout;	Pointer to input/output data structure
Timer *timer;	Pointer to timer

Gauges in the basin

int n_gauges;	Number of gauges
Gauge **output;	Pointer to array of gauges
int *gauge_i;	Array [n_gauges]: column of the gauge
int *gauge_j;	Array [n_gauges]: row of the gauge
Gauge *hyd_gauge;	Extra gauge for 'custom made hydrographs'
int hyd_gauge_i;	Column of extra gauge
int hyd_gauge_j;	Row of extra gauge

Timing data

double dt_rain;	Time step of rain information in min (read) / hour (used)
double lfr_hour;	Time tag of last forecasted rainfall in hours
double lmr_hour;	Time tag of last measured rainfall in hours
double begin_hour;	Time tag of initial time in hours
double inter_hour;	Time tag of intermediate results in hours
double end_hour;	Final hour of every loop

Object methods

Function: *read_env_var(simul)*
Arguments: pointer to Simulator data structure.
Preconditions: nothing
Objective: allocate memory for children data structures and initialize part of the Simulator data structure reading variables from the unix environment.
Return value: void
Side effects: path and parameter variables initialized. Order of computations and soil data are also set.
Algorithm:
get path for geomorphology
compose order of computations file name
open orderfile
read number of pixels
allocate memory for order data structure
read orderfile
get path for state variables

compose root of state variables file name
get path for measured rain
compose root of measured rain file name
compose soil data file name
open soildata file
read number of soil classes
allocate memory for soil data structure
read soildata file
get model parameters

Function: *read_general_data(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data, number of pixels

Objective: initialize timer data structure and allocate memory and initialize basin arrays

Return value: void

Side effects: timer values set. Basin arrays initialized

Algorithm:

```
read timer data
compose stream distance file name
open stream distance file
load stream distance array with raster file data
set pixel geometry data: dx, dy, width, length
start basin
compose gauge distance file name
open gauge distance file
load gauge distance array with raster file data
initialize maximum distance
for all pixels in order array:
  compute distance to outlet
  update maximum distance
compose pointers file name
open pointers file
load pointers array with raster file data
generate basin map
for all pixels in order array: (set 'self' position)
  get row index
  get column index
  get basin array position
  store basin array position in 'self' field of order data structure
for all pixels in order array: (set 'next' position)
  get basin array position of current pixel
  get pointer of current pixel
  get row index of current pixel
  get column index of current pixel
  get row index of downstream pixel
```

get column index of downstream pixel
get basin array position of downstream pixel
store basin array position of downstream pixel in 'next' field of
order data structure
compose slopes file name
open slopes file
load slopes array with raster file data
compose soil type file name
open soil type file
load soil type array with raster file data
get path for initial water table file
compose initial water table file name
open initial water table file
load initial water table array with raster file data

Function: *read_last_nf(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: wetting front file name

Objective: read wetting front file

Return value: void

Side effects: array nf updated

Algorithm:
open wetting front file
load wetting front array with raster data
close file

Function: read_last_nt(simul)

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: top front file name

Objective: read top front file

Return value: void

Side effects: array nt updated

Algorithm:
open top front file
load top front array with raster data
close file

Function: *read_last_d_Mt(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: moisture content file name

Objective: read moisture content file

Return value: void

Side effects: array d_Mt updated

Algorithm:
  open moisture content file
  load moisture content array with raster data
  close file

Function: *read_last_rf(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: runoff generation file name

Objective: read runoff generation file

Return value: void

Side effects: array rf updated

Algorithm:
  open runoff generation file
  load runoff generation array with raster data
  close file

Function: *read_last_hyd(simul,igauge)*

Arguments: pointer to Simulator data structure.
  int: number of gauge

Preconditions: input/output data: path and file names

Objective: read hydrographs

Return value: void

Side effects: result hydrographs updated

Algorithm:
  get gauge baseflow
  open hydro file
  read headings
  until end of file:
   if type tag is one
     read streamflow and rainfall corresponding to previous step
   else
     discard data
  close file

Function: *read_meas_rain(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: measured rainfall file name

Objective: read measured rainfall file

Return value: void

Side effects: array rain updated

Algorithm:
  open measured rainfall file
  load measured rainfall array with raster data
  close file

Function: *read_fore_rain(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: forecasted rainfall file name

Objective: read forecasted rainfall file

Return value: void

Side effects: array rain updated

Algorithm:
  open forecasted rainfall file
  load forecasted rainfall array with raster data
  close file

Function: *read_inter_state(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: file names

Objective: read last state variables

Return value: void

Side effects: arrays nf, nt and d_Mt updated

Algorithm:
  read last wetting front
  read last top front
  read last moisture content
  read last hydrograph

Function: *read_inter_forcing(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: file names

Objective: read rainfall and runoff generation

Return value: void

Side effects: arrays rain and rf updated

Algorithm:
  read last runoff generation
  read last rainfall

Function: *read_initial_state(simul,ref)*

Arguments: pointer to Simulator data structure.

pointer to char: time tag

Preconditions: input/output data: path names

Objective: read initial state variables

Return value: void

Side effects: file names set, result hydrographs and arrays Nf, Nt and d_Mt updated

Algorithm:

```
get time tag
compose file name for wetting front
compose file name for top front
compose file name for moisture content
for all gauges in gauge list:
   compose file name for gauge
read inter state
```

Function: *read_initial_forcing(simul,ref)*

Arguments: pointer to Simulator data structure.

pointer to char: time tag

Preconditions: input/output data: path names

Objective: read initial rainfall and runoff generation

Return value: void

Side effects: file names set and arrays rain and rf updated

Algorithm:

```
get time tag
compose file name for runoff generation
compose file name for measured rainfall
read inter forcing
```

Function: *write_for_hyd(simul,igauge)*

Arguments: pointer to Simulator data structure.

int: number of gauge

Preconditions: input/output data: path and file names, hydrographs

Objective: write hydrograph corresponding to forecasted rain

Return value: void

Side effects: none

Algorithm:

open hydro file
get maximum range in array
for index in array range:
write streamflow and rainfall corresponding to current step

Function: *write_inter_hyd(simul,igauge)*

Arguments: pointer to Simulator data structure.
int: number of gauge

Preconditions: input/output data: path and file names, hydrographs

Objective: write hydrographs corresponding to measured rain

Return value: void

Side effects: none

Algorithm:
open hydro file
get maximum range in array
if forecasted rain is active
write headings for three variables
else
write headings for two variables
for index in array range:
write type tag 0
write streamflow and rainfall corresponding to previous step
for index in array range:
write type tag 1
write streamflow and rainfall corresponding to current step
copy hydro file name as last hydro file

Function: *write_extra_hyd(simul,name)*

Arguments: pointer to Simulator data structure.
pointer to char: file name

Preconditions: input/output data: path and file names, hydrographs

Objective: write hydrographs for the extra gauge. The extra gauge is used to store virtual variables for the user interface. Values stored are not neccessarily hydrographs. Reports of time evolution of virtual variables are also stored here

Return value: void

Side effects: none

Algorithm:
open hydro file
get maximum range in array
write headings for one variable

for index in array range:
write streamflow and rainfall corresponding to current step

Function: *write_inter_sta(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: input/output data: path names, hydrographs and basin state arrays

Objective: write basin state variables and result hydrographs

Return value: void

Side effects: none

Algorithm:
get current time tag
get state variables path
add time tag to state variables path
for all gauges in gauge list:
get hydro path
add time tag to hydro path
compose gauge file name
write hydrograph
compose raster title for wetting front
compose file name for wetting front
open wetting front file
write wetting front array on raster file
close wetting front file
copy wetting front file name as last wetting front file
compose raster title for top front
compose file name for top front
open top front file
write top front array on raster file
close top front file
copy top front file name as last top front file
compose raster title for moisture content
compose file name for moisture content
open moisture file
write moisture content array on raster file
close moisture file
copy moisture file name as last moisture file
compose raster title for runoff generation
compose file name for runoff generation
open runoff file
write runoff generation array on raster file
close runoff file
copy runoff file name as last runoff file

Function: *set_input_list(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: rain paths

Objective: build a list containing all rainfall files in the directory. Rainfall files are identified by root and extension. The list is sorted according to time tag in ascending order.

Return value: void

Side effects: rainlist initialized

Algorithm:

```
set pointers to functions to select and to sort
scan measured rain directory for file names
set number of elements in list
set current element in list to zero
if fore rain tag is yes
   scan forecasted rain directory for file names
   set number of elements in list
   set current element in list to zero
```

Function: *comp_meas(chain)*

Arguments: pointer to direct data structure.

Preconditions: none

Objective: function to select file names in the measured rain directory. It takes the measured rain root and extension and discards items in the directory which do not match.

Return value: void

Side effects: none

Algorithm:

```
get root name
select portion of item name corresponding to root
if compare names is not zero, return zero
select portion of item name corresponding to dot after root
select portion of item name corresponding to colon in time tag
select portion of item name corresponding to dot after time tag
if compare is not zero, return zero
get extension name
select portion of item name corresponding to extension
if compare names is not zero, return zero
return 1
```

Function: *comp_fore(chain)*

Arguments: pointer to direct data structure.

Preconditions: none

Objective: function to select file names in the forecasted rain directory. It takes the measured rain root and extension and discards items in the directory which do not match.

Return value: void

Side effects: none

Algorithm:

```
get root name
 select portion of item name corresponding to root
 if compare names is not zero, return zero
 select portion of item name corresponding to dot after root
 select portion of item name corresponding to colon in time tag
 select portion of item name corresponding to dot after time tag
 if compare is not zero, return zero
 get extension name
 select portion of item name corresponding to extension
 if compare names is not zero, return zero
return 1
```

Function: *order(d1,d2)*

Arguments: pointer to direct data structure: first item to compare
pointer to direct data structure: second item to compare

Preconditions: none

Objective: function to sort file names. It takes the time tag and sorts according to it.

Return value: void

Side effects: none

Algorithm:

```
return comparison of file names
```

Function: *get_next_mrain(simul,mode)*

Arguments: pointer to Simulator data structure.
int: channel for rainfall input: STD_INPUT or AUTO_INPUT

Preconditions: rain paths, measured rain list

Objective: get the next measured file name and evaluate duration of the rainfall loop

Return value: void

Side effects: measured rain file defined, and timer data updated

Algorithm:

```
if mode is STD_INPUT
   read next item in standard input
   if item is STOP return 1
   get measured rain path
   compose measured rain file name
else if mode is AUTO_INPUT
   increment measured list counter
   get next element in list
   get measured rain path
   compose measured rain file name
get time tag of measured rain
if time tag is greater than end time return 1
set parameters for the computation loop: dt_rain and end_hour
```

Function: *get_next_frain(simul,mode)*

Arguments: pointer to Simulator data structure.
int: channel for rainfall input: STD_INPUT or AUTO_INPUT

Preconditions: rain paths, forecasted rain list

Objective: get the next forecasted file name and evaluate duration of the rainfall loop

Return value: void

Side effects: forecasted rain file defined, and timer data updated

Algorithm:

```
if mode is STD_INPUT
   read next item in standard input
   if item is STOP return 1
   get forecasted rain path
   compose forecasted rain file name
else if mode is AUTO_INPUT
   increment forecasted list counter
   get next element in list
   get forecasted rain path
   compose forecasted rain file name
get time tag of forecasted rain
if time tag is greater than end time return 1
set parameters for the computation loop: end_hour
```

Function: *init_gauge(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: number of gauges, path names

Objective: allocate memory and initialize the gauges in the basin

Return value: void

Side effects: gauges initialized

Algorithm:

```
if no gauges defined by the user
  set default gauge
allocate memory for Gauge pointer
allocate memory path names pointers
for all gauges in gauge list:
  allocate memory for the gauge
  get gauge name
  allocate memory for path names
  compose gauge name
  initialize routing map
  allocate memory for results
  initialize results
  if gauge trace is active
    allocate memory for gauge trace
    initialize gauge trace
```

Function: *init_hyd_gauge(simul,i,j)*

Arguments: pointer to Simulator data structure.

int: row index

int: column index

Preconditions: path names

Objective: allocate memory and initialize the extra gauge when it is going to be used for hydrographs. The extra gauge is used by the user interface to store time evolution of virtual variables and user-defined hydrographs

Return value: void

Side effects: extra gauge initialized

Algorithm:

```
if extra gauge is initialized
  free extra gauge
allocate memory for the gauge
compose gauge name
initialize routing map
allocate memory for results
initialize results
```

Function: *init_pix_gauge(simul,i,j)*

Arguments: pointer to Simulator data structure.

int: row index

int: column index

Preconditions: path names

Objective: allocate memory and initialize the extra gauge when it is going to be used for virtual variables. The extra gauge is used by the user interface to store either time evolution of virtual variables or user-defined hydrographs

Return value: void

Side effects: extra gauge initialized

Algorithm:
  if extra gauge is initialized
    free extra gauge
  allocate memory for the gauge
  compose gauge name
  initialize pixel evolution
  allocate memory for results
  initialize results

Function: *start_extra_gauges(simul,ngauges)*

Arguments: pointer to Simulator data structure.
int: number of gauges

Preconditions: nothing

Objective: allocate memory to store location of user-defined gauges

Return value: void

Side effects: memory allocated

Algorithm:
  allocate pointers

Function: *free_hyd_gauge(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: extra gauge initialized

Objective: liberate memory tnd terminate the extra gauge.

Return value: void

Side effects: pointers free

Algorithm:
  free pointers

Function: *sim_loop(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: basin properties, timer data, input/output data

Objective: rainfall and forecasting loops for the whole basin. It computes basin evolution with measured rain, write results if needed and computes basin evolution with forecasted rain if forecasting loop is active.

Return value: void

Side effects: basin state updated, results written. If forecasting loop is active, basin state at the end of the loop does not correspond to the current time, and must be retrieved from the database again if needed.

Algorithm:

```
get timer information and define number of steps
read measured rain
for all steps in dt_rain:
 set initial time of the step
 compute basin velocities
 compute subsurface flow due to lateral heterogeneities
 if step corresponds to forecasted rainfall
   read forecasted rain
 if basin trace is active
    reset basin trace variables
 compute basin evolution (computation loop)
 for all gauges in gauge list:
    compute contribution to the hydrograph
 if basin trace is active
    write basin trace variables
 if step corresponds to end of rain
   if writing results is active
      write basin state and results
```

Function: *hydrograph_loop(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: basin states, timer data, input/output data

Objective: compute hydrograph retrieving basin states from the database

Return value: void

Side effects: results of extra gauge updated. No side effects in timer

Algorithm:

```
get current time
set factors for rain and streamflow
set rainfall list
for all elements in rainfall list:
 get time of next rain file
  if time is earlier than begin time
    discard
  else if time is after current time
    break
```

```
    else
      set timer information
      read measured rain
      read runoff generation
      set initial time of the step
      compute basin velocities
       compute incremental basin response (convolution loop)
```

Function: *pix_loop(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: basin states, timer data, input/output data

Objective: compute hydrograph for one single pixel retrieving basin states from the database

Return value: void

Side effects: results of extra gauge updated. No side effects in timer

Algorithm:

```
  get current time
  set factors for rain and streamflow
  set rainfall list
  for all elements in rainfall list:
   get time of next rain file
    if time is earlier than begin time
      discard
    else if time is after current time
      break
    else
      set timer information
      read measured rain
      read runoff generation
      set initial time of the step
       compute incremental pixel response
```

Function: *rep_loop(simul)*

Arguments: pointer to Simulator data structure.

Preconditions: basin states, timer data, input/output data

Objective: generate report of time evolution of a virtual variable for a pixel retrieving basin states from the database

Return value: void

Side effects: results of extra gauge updated. No side effects in timer

Algorithm:

```
  get current time
  set factors for rain and streamflow
```

```
set rainfall list
for all elements in rainfall list:
 get time of next rain file
   if time is earlier than begin time
     discard
   else if time is after current time
     break
   else
     set timer information
     read measured rain
     read runoff generation
     read basin state
     set initial time of the step
     initialize pixel
     apply virtual function to pixel
     store return value in results of extra gauge
```