BfG-1795

Bericht

HydPy – ein interaktiv nutzbares Framework zur Erstellung und Anwendung hydrologischer Modelle

Autoren:
Dipl.-Umweltwiss. Christoph Tyralla
Prof. Dr. rer. nat. habil. Andreas Schumann

August 2013

Auftraggeber: Bundesanstalt für Gewässerkunde
SAP-Nr.: M39610204022
Anzahl der Seiten: 104

Ruhr-Universität Bochum
Fakultät für Bau- und Umweltingenieurwissenschaften
Lehrstuhl für Hydrologie, Wasserwirtschaft und Umwelttechnik
Prof. Dr. rer. nat. habil. A. Schumann

Impressum

Herausgeber: Bundesanstalt für Gewässerkunde
Am Mainzer Tor 1
Postfach 20 02 53
56002 Koblenz
Tel.: +49 (0)261 1306-0
Fax: +49 (0)261 1306 5302
E-Mail: posteingang@bafg.de
Internet: http://www.bafg.de

DOI: 10.5675/BfG-1795
URL: http://doi.bafg.de/BfG/2013/BfG-1795.pdf

Zitiervorschlag:
TYRALLA, C., SCHUMANN, A. (2013): HydPy – ein interaktiv nutzbares Framework zur Erstellung und Anwendung hydrologischer Modelle, BfG-Bericht 1795, 103 Seiten, Koblenz, doi: 10.5675/BfG-1795.

Inhaltsverzeichnis

1 Modellframework HydPy

1.1 Hintergrund

Die Forschung der Bundesanstalt für Gewässerkunde im Auftrag des Bundesministeriums für Verkehr, Bau und Stadtentwicklung erarbeitet wissenschaftlich fundierte Erkenntnisse als Entscheidungsgrundlagen für das Ressort. Das vorliegende, federführend vom Referat M2 (Wasserhaushalt, Vorhersagen und Prognosen) in Kooperation mit dem deutschen IHP/HWRP-Sekretariat durchzuführende Forschungs- und Entwicklungsvorhaben "Quantifizierung und Reduktion von Unsicherheiten durch Datenassimilation und Ensembletechniken für verkehrsbezogene Kurz-, Mittel- und Langfristvorhersagen der BfG" (Kurztitel "Seamless Prediction") bildet einen Beitrag zum Dachthema "Modelle, Vorhersagen und Auswertungen" des BfG-Forschungskonzepts. Das Vorhaben zielt ab auf die verbesserte Berücksichtigung und gezielte Minderung der vielschichtigen Unsicherheiten in der operationellen Abfluss- und Wasserstandsvorhersage auf sämtlichen verkehrswasserwirtschaftlich relevanten Skalen. Mit innovativen Methoden und Methodenkombinationen sollen die Unsicherheiten (1) durch verbesserte Parametrisierungsstrategien der Modelle sowie Datenassimilationstechniken reduziert, (2) über statistische Verfahren umfassend quantifiziert und (3) mittels einer statistischen Nachbehandlung und Aufbereitung nutzeradäquat kommuniziert werden. Das Gesamtvorhaben gliedert sich in mehrere miteinander vernetzte Teilprojekte, die von der Datenanalyse bis zur Operationalisierung erarbeiteter Methoden reichen. Das Teilprojekt "Unsicherheitsanalyse / Modellunsicherheiten" wird am Lehrstuhl für Hydrologie, Wasserwirtschaft und Umwelttechnik der Ruhr-Universität Bochum bearbeitet. Ziel ist es, die modellinherenten Unsicherheitsquellen (insbesondere Parametrisierung, Modellstruktur, Modellkonzept) für ausgewählte Modelle, Einzugsgebietscharakteristika und Skalen mit unterschiedlichen Methoden / Methodenkombinationen systematisch zu untersuchen. Neben der Unsicherheitsquantifizierung gilt es, bestehende Modellunsicherheiten durch (automatisierte) Optimierung der Parameterwerte zu minimieren und die Prognosefähigkeit der Modelle unter definierten Randbedingungen zu bewerten. Das in diesem Bericht vorgestellte Modellframework stellt ein zentrales Instrumentarium der Unsicherheitsanalyse im Rahmen des F&E-Vorhabens "Seamless Prediction" dar.

1.2 Einführung

HydPy ist ein in Python programmiertes Framework für hydrologische Simulationsrechnungen. Wie andere Frameworks ermöglicht es die Anwendung unterschiedlicher hydrologischer Modelle oder Modellkomponenten in verschiedenen Zeit- und Raumskalen. Die Besonderheit von HydPy ist dessen Flexibilität und Interaktivität. Der erfahrene Anwender kann leicht neue Modelle definieren und einbinden. Zudem kann er dynamisch auf initialisierte Framework- und Modell-Instanzen zugreifen und diese steuern. Daher ist HydPy zur Bearbeitung diverser Problemstellungen sowohl in der Forschung als auch der elaborierten praktischen Anwendung geeignet, ohne dass der Quelltext des Frameworks modifiziert werden muss.

Die erste Entwicklungsphase des Frameworks zielte auf die wissenschaftliche Analyse hydrologischer Modelle hinsichtlich struktureller Unsicherheiten ab. Hierbei standen die Aspekte der raumzeitlichen Variabilität von hydrologischen Prozessen sowie der mathematisch-technischen Umsetzung von Differenzialgleichungen im Vordergrund.

Die räumliche Variabilität hydrologischer Gebietseigenschaften und der hohe Abstraktionsgrad hydrologischer Modelle lassen die Wahl eines einzelnen Modellkonzeptes für alle Teile eines großen Flussgebietes fragwürdig erscheinen. HydPy stellt eine Modell-Basisklasse mit allen notwendigen Schnittstellen zur Verfügung, in die der Anwender die jeweils als prozess- und zweckadäquat angesehenen Modellgleichungen einfügen kann. Die selbst definierten und ebenso die bereits im Framework implementierten Modelle sind beliebig miteinander kombinierbar. Beispielsweise kann im steilen Oberlauf eines Flusses ein anderes Wellenablauf-Verfahren als im flachen Unterlauf angewandt werden. Ebenso können Regionen mit heterogenen Prozessen durch detaillierte Ansätze und feine räumliche Diskretisierungen abgebildet werden, während in homogenen Regionen vereinfachte Ansätze zum Einsatz kommen.

Mathematisch werden hydrologische Prozessgefüge über miteinander gekoppelte Differenzialgleichungen abgebildet. Diese sind grundsätzlich nicht beliebig genau lösbar. HydPy überlässt dem Anwender die Wahl ob modellspezifische „ad hoc“-Lösungsstrategien oder numerische Integrationsalgorithmen verwendet werden sollen. Erstere sind zeiteffizient, führen evtl. aber zu großen Fehlern. Letztere sind vergleichsweise langsam, beschränken dafür aber die numerischen Fehler auf einen vom Anwender vorgegebenen, kleinen Maximalwert. Neben allgemeinen Lösungsalgorithmen für Systeme gewöhnlicher Differenzialgleichungen bietet HydPy darüber hinaus Funktionalitäten um die Differenzialgleichungen selbst hinsichtlich ihrer Lösbarkeit zu optimieren.

Die zweite Entwicklungsphase zielte auf die operationelle Tauglichkeit von HydPy ab. Schwerpunkt war die Erhöhung der Rechengeschwindigkeit, was insbesondere bei der Anwendung numerischer Integrationsalgorithmen relevant ist. In der aktuellen HydPy-Version können laufzeitkritische Simulationsrechnungen nach Fortran ausgelagert werden. Sowohl die Erstellung und Kompilierung des Fortran-Quelltextes als auch das anschließende Einbinden in HydPy geschieht weitgehend automatisch, so dass für die Modell-Erstellung lediglich Python-Kenntnisse notwendig sind. Zum anderen können die Simulationsrechnungen der verschiedenen Modelle vollautomatisch parallel auf mehreren Prozessoren oder – mit etwas Computerkenntnis – auf mehreren Rechnern eines Clusters durchgeführt werden.

In der ausstehenden dritten Entwicklungsphase soll der Komfort für den in Python unerfahrenen Anwender weiter erhöht werden. Dieses betrifft z. B. die Verständlichkeit der Warn- und Fehlermeldungen des Frameworks bei unplausiblen Benutzervorgaben. Zudem werden Methoden zur Verfügung gestellt, welche die automatische Übersetzung von GIS-Daten in HydPy-Steuerdateien erlauben. Darüber hinaus werden weitere Modelle vorimplementiert. Zunächst wird das bereits in mehreren Varianten vorhandene Modell HBV durch LARSIM ergänzt.

HydPy ist auf deterministische „Konzept-Modelle“ der Meso- und Makroskala abgestimmt. „Physikalische-Modelle“, beispielsweise zur detaillierten Hangmodellierung, ließen sich prinzipiell implementieren, dieses wäre allerdings mit großem Aufwand verbunden, da die Framework-Funktionalitäten hierfür nur wenig Unterstützung bieten. Zudem ist grundsätzlich zu bedenken, dass verschiedene Modelle und Teilgebiete voneinander entkoppelt berechnet werden. Rückstaueffekte zwischen Modellen oder Teilgebieten schließen sich somit aus.

Die Interaktivität von HydPy resultiert aus der konsequenten Einhaltung des Paradigmas der Objektorientierung sowie insbesondere aus dem Charakteristikum von Python als dynamische Programmiersprache. Das gesamte Framework ist – entsprechend der Python-Philosophie – wenig vor Benutzerzugriffen abgeschirmt. Auf der obersten Ebene stehen komfortable Funktionen zur Verfügung, die auch vom Einsteiger sicher bedient werden können. Bei Bedarf kann der Fortgeschrittene beliebig weit in die Objekthierarchie eindringen und so die Grundfunktionalitäten von HydPy erweitern.

1.3 Programm-, lizenz- und hardwaretechnische Voraussetzungen

HydPy ist in der Skriptsprache Python programmiert (Python Software Foundation, 2012b). Python wurde Anfang der 90er-Jahre von Guido van Rossum entwickelt. Der Fokus lag und liegt auf leichter Les- und Erlernbarkeit. Dem kommt entgegen, dass Python als dynamische Programmiersprache konzipiert ist, die vergleichbar mit R (The R Foundation, 2006) oder Matlab (MathWorks Deutschland) eine interaktive Ausführung ermöglicht. Aufgrund der großen Flexibilität bei einfacher Syntax wird Python mittlerweile in viele Anwendungen als Skriptsprache herangezogen, beispielsweise im Geoinformationssystem ArcGIS (ESRI Deutschland, 2012). Die lizenzkostenfreie Einbeziehung des Python-Interpreters auch in kommerzielle Anwendungen nach der Python Software Foundation-Lizenz trägt hierzu bei(Python Software Foundation, 2012a).

Python verfügt über eine umfangreiche Standardbibliothek mit weitgehender Plattformunabhängigkeit. Bei der HydPy-Entwicklung wird nach Möglichkeit auf diese Standardbibliothek zurückgegriffen. Zusätzlich kommen die Programmpakete NumPy (Numpy developers, 2012b) und Cython (Cython core developers, 2012) zum Einsatz, die grundlegende Funktionalitäten im Bereich der effizienten Numerik eröffnen und als wissenschaftliche Standardtools gelten. NumPy steht unter der BSD- (Numpy developers, 2012a) und Cython unter der Apache-Lizenz (The Apache Software Foundation, 2012b), die wie die genannte Python Software Foundation-Lizenz freie Software-Lizenzen sind. Da Python-Quelltext zur Laufzeit kompiliert und interpretiert wird, müssen der Python-Interpreter, die Standardbibliothek sowie die genannten Erweiterungen auf dem ausführenden Computer vorhanden sein. Für den Einstieg empfiehlt sich die automatisierte Installation von erweiterten Python-Distributionen wie pythonxy, die auf wissenschaftliche Zwecke ausgerichtet sind (Pythonxy developers, 2012).

Die aktuellen Standardreleases der Referenzimplementierung CPython haben die Versionsnummern 2.7.3 und 3.3.0. HydPy ist in der Version 2.7.3 umgesetzt. Diese ist im Vergleich zur Version 3.3.0 bislang mit mehr Paketen von Drittanbietern kompatibel und wird i. d. R. für wissenschaftliche Zwecke bevorzugt. Innerhalb der Versionslinie 2 besteht Abwärtskompatibilität, so dass HydPy auch unter kommenden Versionen der 2er-Linie lauffähig sein sollte. Zwischen den Versionslinien 2 und 3 besteht keine Abwärtskompatibilität. Die für HydPy relevanten Inkompatibilitäten sind allerdings überschaubar, so dass – sobald die Unterstützung durch externe Pakete das Niveau der 2er-Linie erreicht – der Wechsel in die 3er-Linie in kurzer Zeit möglich ist.

Die oben genannten Stärken von Python gehen mit dem Nachteil einer geringen Geschwindigkeit gegenüber hardwarenahen Programmiersprachen mit statischer Typisierung einher. Zwei Strategien zur Vermeidung langer Rechendauern sind im Framework vorgesehen: (1) HydPy enthält Methoden zum weitgehend automatischen Erstellen, Kompilieren und Einbinden von Fortran-Quelltext entsprechend eigener Python-Routinen. Nach dieser Strategie erfolgt die Modellentwicklung ausschließlich in Python und das resultierende Modell wird anschließend mit vernachlässigbarem Aufwand in eine laufzeiteffiziente Fortran-Version überführt, die als dll-Datei eingebunden wird. (2) Eine ähnliche Vorgehensweise wird durch die Python-Pakete Cython sowie Distutils für C/C++ unterstützt. Hierbei ist der Entwicklungsaufwand aber leicht erhöht, da die betroffenen Python-Routinen manuell mit Informationen zur statischen Typisierung zu versehen sind. In beiden Fällen schränkt die Verwendung plattformspezifischer Fortran- oder C-Compiler die Plattformkompatibilität von HydPy ein. Die laufzeiteffizienten Varianten der bislang umgesetzten Modelle sind lauffähig unter 32- und 64-bit-Versionen von Windows. Falls auf geeignete Compiler zurückgegriffen werden kann, lassen sich die entsprechenden dll-Dateien aber auch für andere Systeme wie LINUX automatisch erstellen.

Hinsichtlich der Hardware bestehen keine klaren Einschränkungen. Bei großen Flussgebieten, einer kleinräumigen Modell-Diskretisierung oder langen Zeitreihen kann es beim Versuch, sämtliche Teilmodelle dauerhaft initialisiert zu halten, leicht zur Überlastung des Arbeitsspeichers kommen. Hierfür wurde zusätzlich die Methode der dynamischen Initialisierung von Modellen implementiert, die auch datenintensive Anwendungsfälle auf 32 bit-Systemen lösbar macht. Allerdings wird eine geringere Laufzeiteffizienz bei wiederholten Simulationen in Kauf genommen. Über die multiprocessing-Routinen wird von mehreren Prozessorkernen eines Rechners Gebrach gemacht. Hierbei ist eine erhöhte Arbeitsspeicher-Grundlast aufgrund ggf. zahlreicher zusätzlicher Prozesse zu berücksichtigen, was auch bei dynamischer Initialisierung zur Überlastung von 32 bit-Systemen führen kann. Die multiprocessing-Routine kann durch geringfügige Erweiterungen für die Nutzung in Rechnerverbunden erweitert werden.

Die Versionsverwaltung des Frameworks erfolgt mit Apache Subversion Subversion (The Apache Software Foundation, 2012a). Diese Versionierungssoftware ist u. a. unter Windows und Linux lauffähig und koordiniert die Speicherung verschiedener Projektversionen. TortoiseSVN erlaubt eine einfache Steuerung von Apache Subversion über den Windows-Explorer (The TortoiseSVN team, 2012). Insbesondere können Änderungen zwischen verschiedenen Arbeitsständen (sogenannte Revisions) leicht nachvollzogen werden.

Neben den speziellen Online-Dokumentationen, die über die oben genannten Quellenangeben zu finden sind, dienen insbesondere die Einführungsbücher von Ernesti & Kaiser (2008) und Vaingast (2009) als Grundlage für die Programmierung des Frameworks HydPy und sind als Einstiegslektüre empfehlenswert.

1.4 Modellframeworks und Objektorientierung

Mit dem Popularitätsgewinn der Objektorientierung steigt die Anzahl der hydrologischen Computerprogramme mit der Bezeichnung Modellframework. Das hydrologische Modellframework erhebt den Anspruch, zur Ausführung verschiedener hydrologischer Modelle in der Lage zu sein. Dessen Programmierung muss damit eine gewisse Allgemeingültigkeit und flexible Schnittstellen aufweisen, was durch das Programmierparadigma der objektorientierten Programmierung erleichtert wird. Prinzipien wie die Zusammenfassung von Attributen und Methoden in Objekten oder die Vererbung von Attributen und Methoden an abgeleitete Klassen sind eine große Hilfestellung, um gut strukturierten, wenig fehleranfälligen und leicht wiederzuverwendenden Quelltext zu schreiben.

Um die Beschreibung des Frameworks HydPy in den folgenden Abschnitten verständlicher zu machen, hier knappe „Definitionen“ und Beispiele für einige Grundbegriffe der objektorientierten Programmierung:

- Klasse: allgemeine Objektbeschreibung (z. B. Fluss)
- Objekt bzw. Instanz: konkrete Umsetzung einer Klasse (z. B. Rhein)
- Attribut: Eigenschaft einer Klasse bzw. eines Objektes (z. B. Länge = 1.230 km)
- Methode: Funktion einer Klasse bzw. eines Objektes (z. B. „bestimme Länge“)
- Member: Sammelbezeichnung für Attribute und Methoden (s. o.)
- Vererbung: Weitergabe der Member einer Klasse an eine abgeleitete Klasse (z. B. Wasserstraße erbt von Fluss Länge sowie „bestimme Länge“ und bekommt zusätzlich das Attribut „Wasserstraßenklasse“)
- Datenkapselung: Verbergen von Membern zur Sicherstellung der Objektintegrität (z. B. Länge kann nicht durch direkten Zugriff, sondern nur über die Methode „neue Länge“ geändert werden, welche prüft ob eine positive reelle Zahl übergeben wird)

1.5 Softwarearchitektur im singleprocessing-Modus

Abbildung 1.1 stellt die zentralen Klassen des Frameworks und ihre Verknüpfungen dar. Nicht dargestellt sind Klassen, die nur im multiprocessing-Modus eine Rolle spielen. HydPy dient als oberste Steuerungseinheit. Anschaulich entspricht eine Instanz der Klasse HydPy beispielsweise einem Flussgebiet. Eingehängt ist jeweils eine Liste von Element- und Node-Instanzen. Jede Element-Instanz enthält die Instanz eines Modells bzw. einer Modellkomponente, welche hydrologische Algorithmen ausführt. Beispielsweise repräsentiert eine Element-Instanz einen Flusslauf und die zugeordnete Model-Instanz das für den Flussabschnitt parametrisierte Muskingum-Verfahren zur Wellenablaufberechnung. Node-Instanzen enthalten keine Model-Instanzen und dienen lediglich der Zusammenführung und Weiterleitung von Flüssen. Eine Node-Instanz entspricht beispielsweise einer Pegelmessstelle. Die Klassen ParameterHandler und ArrayHandler bieten verschiedene Funktionalitäten für den Umgang mit Modellparametern und Zeitreihen-Matrizen. Sie müssen nicht, sollten aber zwecks Vereinheitlichung von jeder dem Framework hinzugefügten Model-Klasse verwendet werden. Die Unterklasse InputHandler bietet Funktionalitäten zum Einlesen von Zeitreihen. Eine FluxOrStorageHandler-Instanz steuert das Initialisieren, Rücksetzen und Ausschreiben der interner Fluss- oder Speicherarrays sowie die Anfangswertbelegung der Speicherarrays.

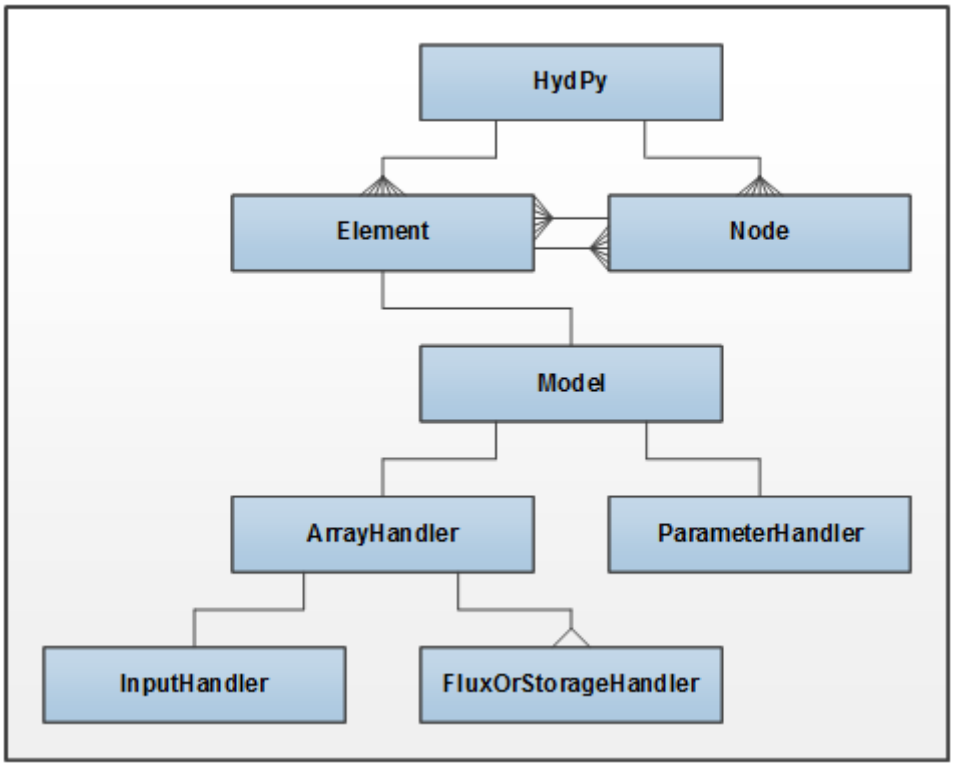


Abbildung 1.1: Objektstruktur im singleprocessing-Modus

Die Klasse HydPy stellt den anwendungsüblichen Funktionsumfang bereit. In der Regel müssen nur wenige HWU-Member bekannt sein. Die Initialisierung, Verknüpfung und Steuerung der nachgeschalteten Objekte erfolgt automatisiert durch die HydPy-Instanz. Jedoch kann von der HydPy-Instanz ausgehend jederzeit ein direkter Zugriff auf alle nachgeschalteten Objekte und deren Member erfolgen. Diese Verletzung des Prinzips der in Python ohnehin nur bedingt möglichen Datenkapselung kann leicht zu nicht vorgesehenen Fehlern führen. Da Fehler in aller Regel aber nicht im Programmabsturz, sondern in der Rückgabe weitgehend verständlicher Fehlermeldungen münden und – entsprechend der Python-Philosophie – von einem kompetenten und gutwilligen Nutzer ausgegangen wird, wird diese Einschränkung zugunsten des Flexibilitätsgewinns in Kauf genommen.

1.6 Vernetzung von Element- und Node-Instanzen

Die Element- und Node-Klassen ermöglichen vielfältige Vernetzungsmöglichkeiten von Modellen und Modellkomponenten. An Restriktionen ist zu beachten, dass (1) nur Verknüpfungen vom Typ Node-Element bzw. Element-Node, nicht aber vom Typ Node-Node oder Element-Element erlaubt sind (2) alle Flüsse unidirektional sind, Flüsse also ausschließlich vom Ober- zum Unterlieger weitergegeben werden, (3) der Unterlieger keinen Einfluss auf den Zufluss des Oberliegers hat, was beispielsweise Rückstaueffekte zwischen in getrennten Elementen simulierten Flussabschnitten ausschließt, (4) Zirkelverbindungen nicht erlaubt sind, (5) jede Node-Instanz nur für eine Fluss-Komponente zuständig ist.

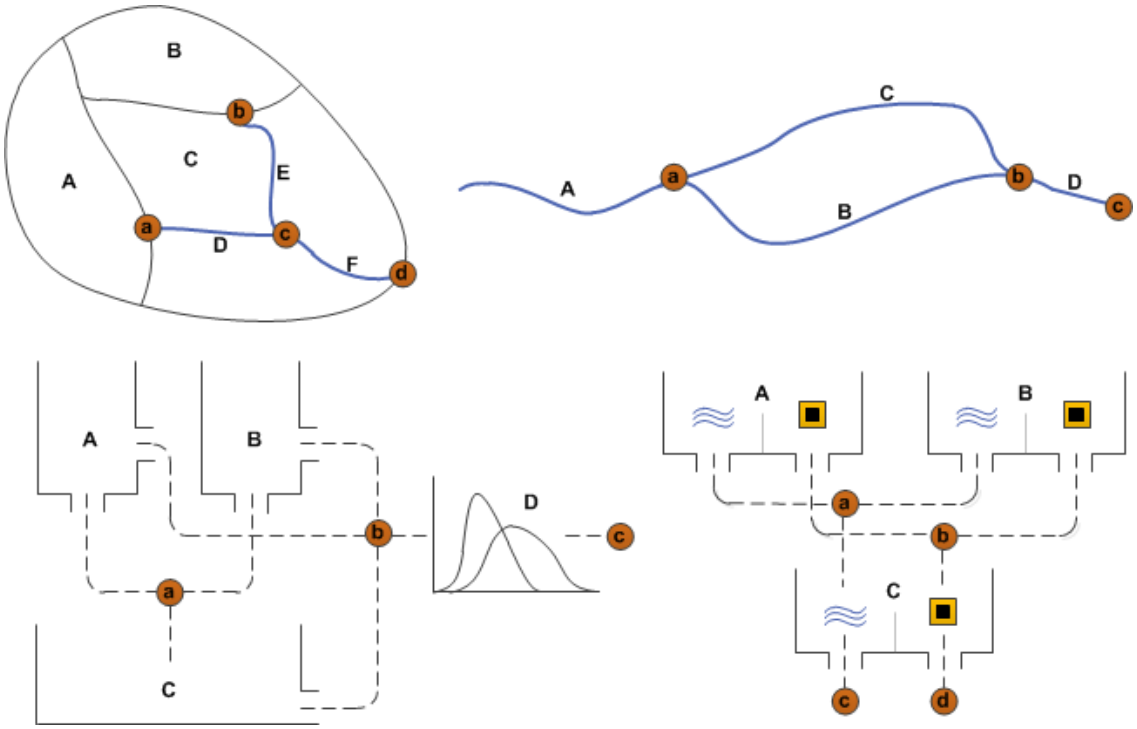


Abbildung 1.2: Mögliche Vernetzungen der Element- (Großbuchstaben) und Node-Instanzen (Kleinbuchstaben)

Abbildung 1.2 veranschaulicht vier Fälle, die typische Vernetzungen von unidirektionalen hydrologischen Modellen und Modellkomponenten weitgehend abdecken. Alle mit Großbuchstaben benannten Objekte stellen Element-Instanzen dar. Node-Instanzen sind grundsätzlich als Kreise dargestellt und mit Kleinbuchstaben benannt.

Oben links ist ein Flussgebiet in die drei Teilgebiete A, B und C unterteilt. Sie entwässern in die Verbindungselemente a, b und d. Die Flussabschnitte D und E nehmen den Abfluss von a und b auf, führen ihn bei c zusammen, von wo er von F an d weitergereicht und somit zum Teilgebietsabfluss von C addiert wird.

Oben rechts sind lediglich Flussabschnitte und deren Verbindungselement dargestellt. Der Zufluss aus A wird von a sowohl an B als auch an C weitergereicht. Da Node-Instanzen nur Aussummieren und Weiterleiten, müssen die Flussabschnitte B und C die Erhaltung der Wasserbilanz sicherstellen, d. h. die für sie bestimmten Teilwassermengen extrahieren.

Unten links übernehmen dagegen die „sendenden“ Element-Instanzen A und B die Aufteilung des Flusses. A, B, C und D repräsentieren mehrere Modellkomponenten, die ihrerseits das Abflussgeschehen beispielsweise eines Teilgebietes repräsentieren. A und B teilen ihre Ausgabe in eine langsame und eine schnelle Abflusskomponente auf und reichen diese an a und b weiter. Die langsame Komponente wird durch C verzögert und anschließend zur schnellen Abflusskomponente in b addiert. Die Summe der Komponenten geht dann in ein Abflusskonzentrationsverfahren ein.

Unten rechts werden verschiedene Flussarten vom den Element-Instanzen A und B an C weitergereicht. Gleichgültig, ob es sich um verschiedene Abflusskomponenten oder um die Kombination von Abfluss- und Stofffrachten handelt, ist jede Flussart über jeweils eine Node-Instanz abzuwickeln. Es besteht keine Einschränkung hinsichtlich der Art der weitergeleiteten Flüsse, da das Framework selbst nur deren Steuerung übernimmt. Allerdings muss die Ausgabe einer Flussart von allen betroffenen Element-Instanzen in derselben Einheit erfolgen.

Bislang zielt der HydPy-Entwicklungsprozess primär auf die beiden oberen Vernetzungstypen ab, die eher Modelle als Modellkomponenten kombinieren. Daraus folgt insbesondere, dass keine Interaktion zwischen Element-Instanzen möglich ist. Jede Model-Instanz muss, vom weitergereichten Zufluss abgesehen, vollkommen unabhängig von den anderen Model-Instanzen lösbar sein. Beispielsweise wäre die Lösung der Richards-Gleichung zur Berechnung der Wasserbewegung im Untergrund mit der Finite-Elemente-Methode im Framework prinzipiell nur möglich, wenn alle finiten Elemente in einer Model-Instanz vorliegen.

1.7 Softwarearchitektur im multiprocessing-Modus

Abbildung 1.3 zeigt die zentralen Klassen des Frameworks in ihrer Vernetzung im multiprocessing-Modus. Um die Berechnungen auf mehrere Prozessorkerne aufzuteilen, werden separate Prozesse gestartet. Die HydPy-Instanz sowie sämtliche Element- und Node-Instanzen werden in jeweils in einem eigenständigen Prozess initialisiert und gehalten. Die Kommunikation zwischen diesen erfolgt entsprechend dem Client-Server-Modell. Im singleprocessing-Modus verfügt eine HydPy-Instanz über eine Referenz auf eine Element-Instanz, um Befehle wie den Simulationsstart weiterzugeben; und eine Element-Instanz über eine Referenz auf eine Node-Instanz, um die ermittelte Ausgabe weiterzureichen. Da eine solche direkte Kommunikation zwischen separaten Prozessen nicht möglich ist, werden auf HWU-Seite ElementClient- und NodeClient-Instanzen zwischengeschaltet. Diese bauen über IP-Adresse und Port-Nummer eine Netzwerkverbindung zu einer ElementServer- bzw. NodeServer-Instanz auf. ElementServer- und NodeServer-Instanz stellen dabei Proxys zur Verfügung, welche die Steuerung der Element- und Node-Instanzen ermöglichen. Zudem kann jede ElementServer- oder NodeServer-Instanz selbst als Client an die unterhalb liegende NodeServer- bzw. ElementServer-Instanz angebunden sein. Modellausgaben werden auf diesem direkten Weg über Queues („Warteschlangen“) weitergereicht.

Von der erhöhten Rechengeschwindigkeit abgesehen, ergibt sich im typischen Anwendungsfall für den Benutzer kein Unterschied in der Handhabung des Frameworks. Die ElementClient- und NodeClient-Klassen emulieren die Element- und Node-Klassen dahingehend, dass der Benutzer ausgewählte Methoden der ersteren so nutzen kann, als ob er direkt auf letztere zugreifen würde. Allerdings sollte nach der Modellnutzung und insbesondere im Störungsfall das korrekte Schließen sämtlicher Prozesse sichergestellt werden, worauf im Abschnitt 1.12.5 eingegangen wird.

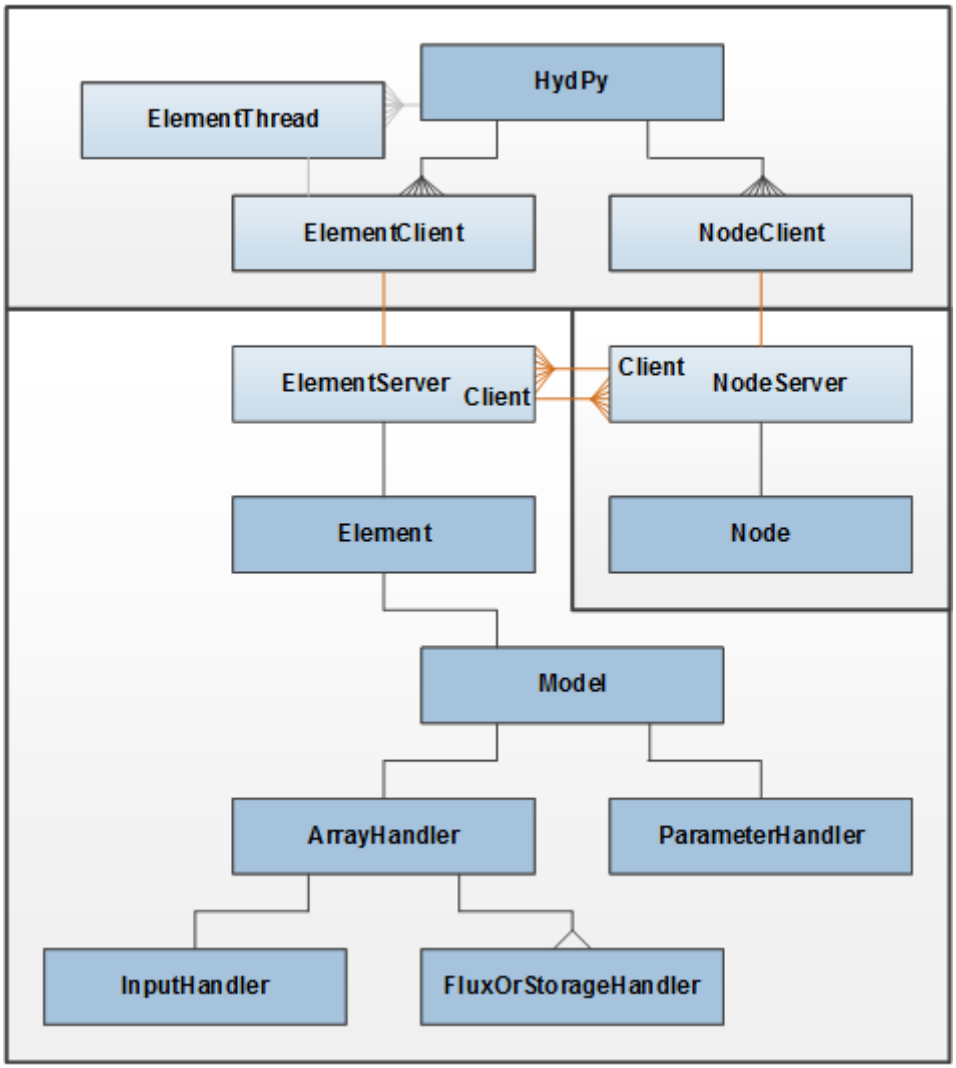


Abbildung 1.3: Objektstruktur im multiprocessing-Modus

1.8 Verzeichnisstruktur

Abbildung 1.4 zeigt Ordnerhierarchie und Beispieldateien für das Projekt *rhein*. Der Quelltext befindet sich in der Datei HydPy.*py* sowie den Ordnern *framework*, *addIns* und *models*. Dieser ist für alle Projekte identisch. Entsprechend ist zum Ende der Entwicklungsphase eine Ausgliederung in die Python-Bibliothek geplant. Die Ordner *controlFiles*, *initialValueFiles* und *timeSeries* enthalten dagegen projektspezifische Informationen. Auf *temp* greift HydPy zur Zwischenspeicherung nur temporär und frameworkintern relevanter Daten zu.

Mit Import des Moduls HydPy.*py* werden sämtliche übrigen frameworkin- und –externen Module geladen bzw. die entsprechenden System-Pfade hierfür dem Python-Interpreter bekannt gemacht. Im Ordner *framework* sind die in den Abschnitten 1.5 und 1.7 beschriebenen zentralen Objektdefinitionen in getrennten Modulen enthalten. Ausgenommen hiervon sind die unter *models* abgelegten hydrologischen Modelle. Grundsätzlich muss jede Model-Klasse in einem Modul im Unterordner *python* definiert sein. Im Beispiel ist das auf zwei Komponenten aufgeteilte

Modell HBV_{96} (siehe Abschnitt 2.1) in den Modulen *HBV96_zone.py* und *HBV96_stream.py* enthalten. Unterordner mit vorangestelltem *c* oder *f* enthalten aus kompiliertem C- oder Fortran-Quelltext hervorgegangene Python-Module. Der weitere Ordnername benennt das der Kompilierung zugrunde liegende Betriebssystem. Module in diesen Ordnern enthalten in i. d. R. nicht die vollständige Model-Klasse, sondern lediglich optimierte Versionen der laufzeitkritischen Methoden zur Durchführung der eigentlichen Simulation. Diese werden von der im Unterordner *python* definierten vollständigen Model-Klasse gesteuert. Im Unterordner *cython* hinterlegt ist der noch nicht übersetzte und kompilierte Cython-Quelltext des jeweiligen Modells sowie in *Setup.py* die plattformunabhängigen Kompilier- und Linkanweisungen. Diese sind lediglich notwendig, wenn die Cython-Implementierung eines Modells unter einem bis dahin noch nicht berücksichtigten Betriebssystem eingesetzt werden soll.

Die Steuerdateien eines HydPy-Projektes sind im Ordner *controlFiles* gespeichert. Deren Inhalte sind detailliert im Abschnitt 1.9 beschrieben.

Der Name der Hauptsteuerdatei ausschließlich der Endung *.main* dient als Projektname. Dieser findet standardmäßig in verschiedenen Datei- und Ordnernamen der Verzeichnisstruktur Verwendung. Allerdings können alternative Namen in der Hauptsteuerdatei festgelegt werden. Ansonsten enthält die Hauptsteuerdatei grundsätzliche Einstellungen wie beispielsweise den Starttermin des Simulationszeitraumes. In einer Netzwerkdatei mit der Endung *.network* wird die Vernetzung aller Element- und Node-Instanzen festgelegt. Parameter-Dateien mit der Endung *.contrPars* enthalten Angaben zur Initialisierung des hydrologischen Modells einer Element-Instanz.

In Unterordnern von *initialValueFiles* werden die initialen Zustände der hydrologischen Modelle in Anfangswert-Dateien mit der Endung *.initVals* beschrieben. Jeder Element-Instanz ist jeweils eine Datei zugeordnet. Um Anfangsbedingungen für verschiedene Startzeitpunkte hinterlegen zu können, sind mehrere Unterordner anzulegen. Deren Namen bestehen standardmäßig aus dem Datum des Startzeitpunktes. Durch die Hauptsteuerdatei oder während der interaktiven Steuerung von HydPy (siehe Abschnitt 1.12.2) können alternative Namen festgelegt werden.

Zeitreihen werden nach Ein- und Ausgabedaten differenziert in *timeSeries* hinterlegt. Je nachdem ob die Daten im ASCII-Format oder einem NumPy-internen Binärformat vorliegen, werden sie in Unterordnern von *asc* bzw. *npy* gespeichert. Bei den Ausgabedaten kann zusätzlich auf ein weiteres, auf die Speicherung mehrerer Simulationsläufe abzielendes Binärformat mit der Endung *bin* zurückgegriffen werden. Die Namen der modellbezogenen Dateien enthalten nach dem letzten Unterstrich den in der Model-Klasse definierten Namen der jeweiligen Eingangsgröße. Das Modell der ersten Element-Instanz liest somit u. a. Werte der Temperatur *T* und des Niederschlages *P* ein und schreibt berechnete Werte des Abflusses *Q* aus. Node-Instanzen enthalten nur eine Flussart, daher das grundsätzliche Kürzel *sim* für ausgeschriebene Werte. Das Kürzel *obs* steht für eine Messreihe, die von einer Node-Instanz optional eingelesen werden kann. Die Datei *info.txt* enthält allgemeine Informationen über die im selben Ordner abgelegten Dateien. Das Framework greift auf diese zurück und prüft auf Überstimmung mit den Angaben der Hauptsteuerdatei. Details zu Zeitreihen-Dateien finden sich im Abschnitt 1.10.

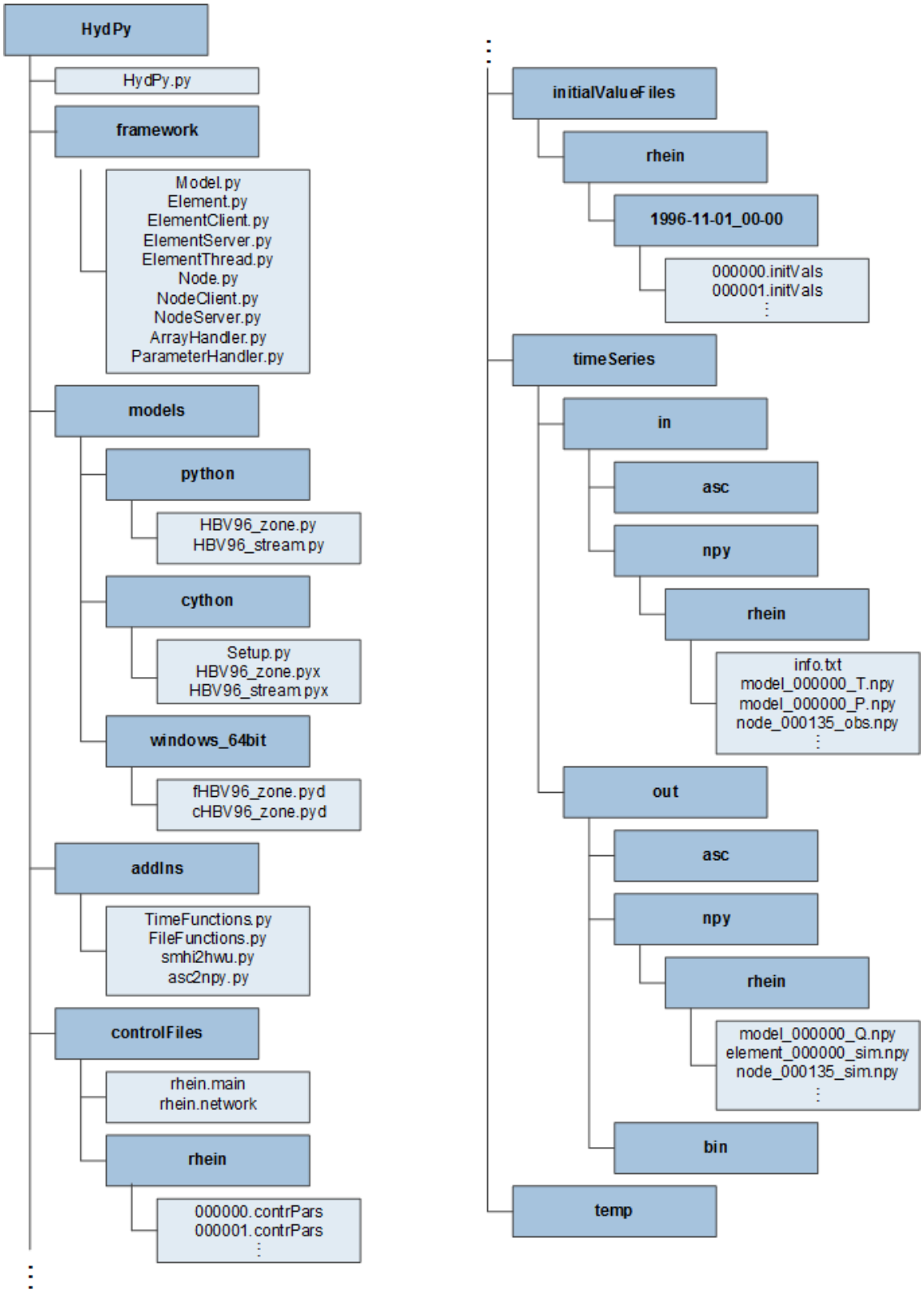


Abbildung 1.4: Projektstruktur mit Beispieldateien

1.9 Steuerdateien

In sämtlichen Steuerdateien sind die Schlüsselwörter links durch Tabulator-Zeichen von den Werten rechts getrennt. In bestimmten Daten kann Einrücken durch Tabulator-Zeichen am Zeilenanfang – wie in Python-Quelltext – zur Zusammenfassung von Textblöcken dienen. Die Reihenfolge von nicht durch Einrücken verbundenen Zeilen ist beliebig. Das Verwenden von Leerzeichen ist i. d. R. nicht erlaubt; das von nicht in der ASCII-Kodierung enthaltenen Zeichen nicht empfohlen. Mit einem Doppelkreuz (#) beginnende Zeilen werden als Kommentar interpretiert und ebenso wie Leerzeilen beim Einlesen ignoriert.

1.9.1 Hauptsteuerdatei (.main)

In Abbildung 1.5 sind sämtliche möglichen Schlüsselwörter der Hauptsteuerdatei mit einer kurzen Erklärung aufgeführt. Optionale Schlüsselwörter müssen nicht in der Datei enthalten sein. Für diese ist die Standardbelegung in Klammern angegeben. Einige Angaben der Hauptsteuerdatei wie z. B. das Datenformat dürfen in interaktiven Modus nachträglich geändert werden. Dies hat keinen einen Einfluss auf den Inhalt der Steuerdatei. Die Benennung der Schlüsselwörter entspricht weitgehend derjenigen der Variablen im Python-Quelltext. Die Verwendung von Klein- und Großbuchstaben in den Schlüsselwörtern ist beliebig.

file_network	Name der Netzwerk-Datei ohne Endung (Projektname)
folder_contrPars	Ordnername der Parameter-Dateien (Projektname)
folder_initVals	Ordnername der Anfangswert-Dateien (Projektname)
folder_timeSeriesIn	Ordnername der Eingangsdateien (Projektname)
folder_timeSeriesOut	Ordnername der Ausgangsdateien (Projektname)
subfolder_initVals	Unterordner der Anfangswert-Dateien (date_startSim)
mode_timeSeriesIn	Format der Eingangsdaten: asc oder npy (npy)
mode_timeSeriesOut	Format der Ausgangsdaten: asc, npy oder bin (npy)
number_nodes	Anzahl der Node-Instanzen
number_elements	Anzahl der in der Element-Instanzen
timestep	Zeitschrittweite der Simulation und der Daten
date_startData	Zeitpunkt, ab dem sämtliche Datenreihen beginnen
date_endData	Zeitpunkt, an dem sämtliche Datenreihen enden
date_startSim	Zeitpunkt, an dem die Simulation startet (data_startData)
date_endSim	Zeitpunkt, an dem die Simulation endet (data_endData)

Abbildung 1.5: Beschreibung Hauptsteuerdatei

Der Name der Hauptsteuerdatei entspricht dem Projektnamen plus der Endung *.main*. Soweit nicht anders angegeben, werden alle projektspezifischen (Ober-)Ordner mit dem Projektnamen benannt. Der Name des Unterordners der Anfangswert-Dateien basiert standardmäßig auf dem

Startdatum der Simulation. Ohne Pfadangaben gelten die Abbildung 1.4 dargestellten Ordner-Positionen in der Verzeichnisstruktur. Pfadangaben sind relativ zu diesen Positionen möglich. Als Trennzeichen für verschiedene Ordner in der Pfadangabe kann „>“ verwendet werden. Dieses wird frameworkintern durch das Trennzeichen des jeweiligen Betriebssystems ersetzt.

Die Zeitschrittweite der Eingangsdaten muss mit derjenigen der Simulation identisch sein und wird dem Framework über das Schlüsselwort *timestep* bekannt gemacht. Schrittweiten in vollen Minuten, Stunden oder Tagen sind zulässig. Dafür wird eine ganze Zahl direkt der gewünschten Zeiteinheit vorangestellt. *5m* entspricht beispielsweise fünf Minuten, *1h* einer Stunde und *30d* dreißig Tagen.

Start- und Endzeitpunkte sind im Format *YYYY-MM-DD_hh:mm* anzugeben. Dem 24.12.2012, 16:15 Uhr entspricht der String *2012-12-24_16:15*. Diese Konvention gilt grundsätzlich im Framework. Ausnahme ist der i. d. R. aus einem Datum gebildete Name des Unterordners der Anfangswerte. Da der Doppelpunkt in Pfadangaben nicht zulässig ist, wird er durch einen weiteren Bindestrich ersetzt. Die Angabe in der Hauptsteuerdatei kann aber dennoch mit einem Doppelpunkt erfolgen.

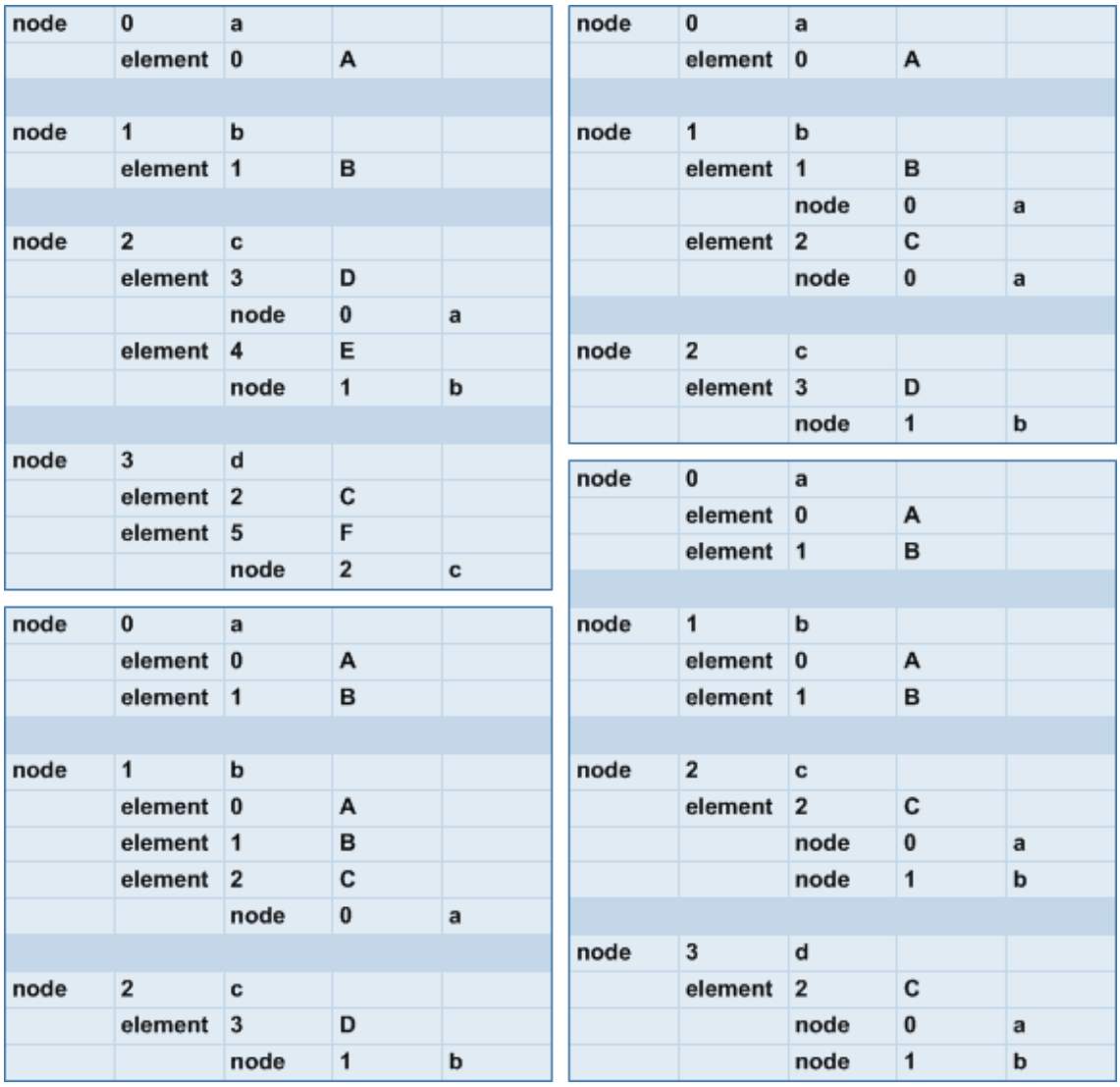
1.9.2 Netzwerkdatei (.network)

In der Netzwerkdatei werden Verbindungen zwischen Node- und Element-Instanzen angegeben und so ein Flussnetzwerk definiert. Ihr Name setzt sich aus – falls in der Hauptsteuerdatei nicht anders angegeben – dem Projektnamen sowie der Endung *.network* zusammen. In Abbildung 1.2 sind mögliche Vernetzungen anhand von vier Beispielen veranschaulicht. In identischer Anordnung zeigt Abbildung 1.6 den Inhalt der entsprechenden Netzwerkdateien.

Jede Zeile setzt sich aus dem Schlüsselwort *node* oder *element*, einem laufenden Index sowie dem Namen der Instanz zusammen. Verbindungen zwischen Node- und Element-Instanzen werden über Einrücken mit dem Tabulator-Steuerzeichen am Zeilenanfang angegeben. Zudem erlaubt sind Leerzeilen zur Strukturierung und mit dem Doppelkreuz beginnende Zeilen zur Kommentierung.

Durch Einrücken verbundene Zeilen werden im Folgenden als Block bezeichnet und sind hellblau dargestellt. Da die optimale Rechenreihenfolge der verschiedenen Node- und Element-Instanzen bei der Modellinitialisierung algorithmisch bestimmt wird (siehe Abschnitt 1.12.5), kann die Reihenfolge der Blöcke willkürlich gewählt werden. Jeder Block beginnt mit einer Zeile einer Node-Instanz. Gefolgt wird diese Zeile von wenigstens einer weiteren. Diese ist durch ein Tabulatorzeichen eingerückt und beschreibt eine oberhalb liegende Element-Instanz. Sollte die Element-Instanz ihrerseits Zuflüsse erhalten, wird dies über wenigstens eine direkt folgende Zeile mit doppeltem Tabulatorzeichen angegeben.

Node- und Element-Instanzen erhalten eigene laufende Indices. Diese müssen jeweils bei null beginnen. Die Reihenfolge des Auftretens der Indices in der Netzwerkdatei ist willkürlich. Namen können beliebig vergeben werden. Allerdings sollte insbesondere bei Node-Instanzen auf Dopplungen verzichtet werden, sonst ist der vom Framework unterstützte Zugriff auf Instanzen über deren Namen nicht möglich (siehe Abschnitt 1.12.6).



node	0	a		
	element	0	A	
node	1	b		
	element	1	B	
node	2	c		
	element	3	D	
		node	0	a
	element	4	E	
		node	1	b
node	3	d		
	element	2	C	
	element	5	F	
		node	2	c

node	0	a		
	element	0	A	
node	1	b		
	element	1	B	
		node	0	a
	element	2	C	
		node	0	a
node	2	c		
	element	3	D	
		node	1	b

node	0	a		
	element	0	A	
	element	1	B	
node	1	b		
	element	0	A	
	element	1	B	
node	2	c		
	element	2	C	
		node	0	a
		node	1	b
node	3	d		
	element	2	C	
		node	0	a
		node	1	b

node	0	a		
	element	0	A	
	element	1	B	
node	1	b		
	element	0	A	
	element	1	B	
node	2	c		
	element	2	C	
		node	0	a
		node	1	b
node	3	d		
	element	2	C	
		node	0	a
		node	1	b

Abbildung 1.6: Steuerdateien der in Abbildung 1.2 dargestellten Vernetzungen.

Eine Besonderheit ergibt sich für die in Abbildung 1.2 und Abbildung 1.6 unten rechts dargestellte Vernetzung. Hier werden zwei Flussarten simuliert. In solchen Fällen legt die Reihenfolge des Auftretens in der Netzwerkdatei fest, welche Node-Instanz welche Flussart übernimmt. Jede Model-Klasse definiert die Reihenfolge ihrer Flussarten, falls mehrere vorliegen. Im Beispiel übernimmt *node a* die erste Flussart von *element A* und *element B*, *node b* hingegen die zweite (definiert in den ersten beiden Blöcke). *element c* übernimmt die Werte von *node a* als erste Flussart und die von *node b* als zweite (redundant definiert in den letzten beiden Blöcken).

1.9.3 Modell-Initialisierungsdatei (.contrPars)

Modell-Initialisierungsdateien enthalten Informationen über die von den Element-Instanzen zu verwendenden Model-Klassen sowie zu deren Initialisierung. Jeder Element- und somit jeder Model-Instanz ist eine Modell-Initialisierungsdatei zugeordnet. Deren Name lautet *xxxxxx.contrPars*, mit *xxxxxx* als Platzhalter für die ID bzw. den laufenden Index der Element-Instanz mit vorangestellten Nullen. Leer- und Kommentarzeilen sind an jeder Stelle zulässig.

In jeder Modell-Initialisierungsdatei ist das Schlüsselwort *mode* enthalten, gefolgt von der durch die Element-Instanz zu instanziierenden Model-Klasse. Deren Bezeichnung muss – auch in ihrer Klein- und Großschreibung – mit dem Namen des entsprechenden Python-Moduls sowie dem Namen der darin enthaltenen Klassendefinition identisch sein. Alle weiteren Angaben sind modellspezifisch und werden vom Framework an den Konstruktor der Model-Instanz übergeben.

In Abbildung 1.7 werden mehrere Formatierungskonventionen veranschaulicht. Diese sind teils erforderlich um ein sicheres Einlesen zu gewährleisten, teils empfehlenswert um weitere HydPy-Funktionalitäten nutzen zu können und beispielsweise die Parameterhandhabung nicht für jede Model-Klasse neu definieren zu müssen. Welche konkreten Angaben eine bestehende Model-Klasse in welcher Form benötigt, ist der jeweiligen Klassendokumentation zu entnehmen.

mode	HBV96_zone		
area	123.4		
unitTimestep	1d		
MAXBAZ	0.0		
ABSTR	nan		
PCORR	1.0,1.1,0.95		
PCORR	1.0 1.1 0.95		
PCORR	1.0	1.1	0.95
ratingCurve	list		
	0,1,2		
	0	1	3
ratingCurves	dict		
	H	0.0,1.0,2.0	
	Q	dict	
		old	0,1,3
		new	0,1,4
ratingCurveFile	rcf.bin		

Abbildung 1.7: Formatierungskonventionen für Modell-Initialisierungsdateien

Im einfachsten Fall wird ein Schlüsselwort samt der zugeordneten Information in einer Zeile angegeben, wobei ein Tabulatur-Zeichen zur Trennung dient. Sowohl Schlüssel als auch Information werden im Framework als String interpretiert, womit beliebige ASCII-Zeichen zulässig sind. Die durch die Model-Klasse definierten Groß- und Kleinschreibungen sind zu beachten. Als Dezimaltrennzeichen dient der Punkt.

Die stets erforderliche Angabe der Model-Klasse in der ersten Zeile wurde bereits beschrieben. Dem nächsten Schlüssel *area* ist der Wert *123.4* zugeordnet. Sofern in der Klassendokumentation nicht anders angegeben, gilt für Gebietsgrößen die Einheit km². *unitTimestep* gibt an, auf welchen Zeitraum sich die angegebenen Werte der zeitabhängigen Parameter beziehen. Parame-

terwerte werden modellintern in die Zeitschrittweite der Simulation überführt. Diese wird über formatgleiche Angaben in der Hauptsteuerdatei definiert (siehe Abschnitt 1.9.1).

MAXBAZ, *ABSTR* sind zwei HBV_{96}-spezifische Modellparameter. *MAXBAZ* wird mit dem Wert *0* belegt. Parameterwerte werden unabhängig von einer Schreibweise mit oder ohne Dezimaltrennzeichen in der Regel als Gleitkommazahlen gespeichert. Für *ABSTR* soll kein Zahlenwert übergeben werden, was durch das Kürzel *nan* (not a number) kenntlich gemacht ist.

PCORR ist ein Parameter, der für jede der (im Beispiel 3) Untereinheiten eines HBV_{96}-Modells getrennt vorliegt. In derartigen Fällen können mehrere Werte durch Kommas sowie einfache Leer- oder Tabulatorzeichen getrennt übergeben werden. Frameworkintern werden Leer- und Tabulatorzeichen durch Kommas ersetzt.

Anhand von Abflusskurven ist veranschaulicht, wie Informationen mit tieferer Struktur anzugeben sind. Im ersten Fall ist eine einfache Schlüsselkurve definiert. Durch den reservierten Eintrag *list* hinter *ratingCurve* werden die folgenden gleich eingerückten Einträge als Liste zusammengefasst an die Model-Instanz weitergereicht. In diesem Fall müsste der Model-Instanz die Reihenfolge der Wasserstand- und der Abflusswerte bekannt sein. Tiefere Verschachtelungen mit Benennung ermöglicht der reservierte Eintrag *dic*. Unter *dic* werden bei gleicher Einrücktiefe weitere Unterschlüsselwörter eingeführt. Im konstruierten Beispiel gibt es nur eine Variante der Wassertiefen, aber zwei des Abflusses.

Erfordert ein Modell Informationen, die durch die Formatierungskonventionen der Modell-Initialisierungsdatei nicht befriedigend erfasst sind, können Pfadangaben zu ergänzenden Dateien aufgenommen werden. Das „>“-Zeichen kann wie in der Hauptsteuerdatei als vom Betriebssystem unabhängiges Trennzeichen genutzt werden. Im Beispiel würde der Model-Instanz über das Schlüsselwort *ratingCurveFile* ein entsprechender Pfad- bzw. Dateiname übergeben.

Sowohl die über die Einträge *list* und *dic* eingelesenen Informationen mit größerer Informationstiefe als auch diejenige aus separaten Dateien werden durch die ParameterHandler-Klasse nicht automatisch unterstützt. Der Rückgriff auf diese Möglichkeiten der Datenbereitstellung geht daher mit größerem Programmieraufwand für die jeweilige Model-Klasse einher. Gleiches gilt natürlich für die individuelle Einleseroutine eines alternativen Dateiformats selbst.

Sämtliche Schlüssel-Information-Paare werden an den Konstruktor der Model-Klasse weitergereicht. Die Model-Klasse fragt nur die für sie relevanten Informationen ab; überflüssige Dateiinhalte werden nicht beachtet und führen somit nicht zu Fehlern. Eine Besonderheit sind die System-Anfangszustände. Sind diese in der Datei enthalten, werden sie bei der Modell-Initialisierung verwendet. Da aber keine zeitbezogene Differenzierung der Anfangszustände in der Modell-Initialisierungsdatei möglich ist, sollte i. d. R. auf die hierfür vorgesehenen Anfangswertdateien zurückgegriffen werden. Allgemein gilt, dass bei einer Mehrfachnennung desselben Schlüssels bei abweichenden Informationen eine Fehlermeldung erfolgt.

1.9.4 Anfangswertdatei (.initVals)

Anfangswertdateien bilden den Systemzustand eines Modells zu einem bestimmten Zeitpunkt ab, so dass von diesem ausgehend „weitergerechnet“ werden kann. Für jede Element- bzw. Model-Instanz muss eine Datei vorliegen. Die Benennung erfolgt wie bei Modell-Initialisierungsdateien, allerdings mit der Endung *.initVals*. Die Formatierungskonventionen

sind identisch.

Üblicherweise sollte von einzeiligen Schlüssel-Information-Angaben Gebrauch gemacht werden. Der Schlüssel entspricht dann der modellinternen Bezeichnung der Speichergröße, die Information wird über die in die Model-Instanz eingehängte ArrayHandler-Instanz automatisch zur Setzung des Anfangswertes bzw. der Anfangswerte des entsprechenden Arrays verwendet. Nur übergebene Anfangszustände werden gesetzt bzw. verändert.

Für die komplexeren Formatierungen bestehen die im vorherigen Abschnitt besprochenen zusätzlichen Erfordernisse an die Model-Klasse.

1.10 Zeitreihen-Dateien

1.10.1 Zeitreihen-Eingabeformate (.asc, .npy)

Jeder Model-Instanz sind – der Erfordernis des Modellkonzeptes entsprechend – beliebig viele Dateien mit Eingangsdaten zugeordnet. Deren Benennung erfolgt entsprechend dem Format *model_xxxxxx_X.asc* bzw. *model_xxxxxx_X.npy*. *xxxxxx* ist Platzhalter für die ID bzw. den laufenden Index der die Model-Instanz enthaltenden Element-Instanz, die durch vorangestellte Nullen sechsstellig einzutragen ist. *X* ist Platzhalter für den modellinternen Namen der Eingangsgröße. Die Einleseroutine der ArrayHandler-Klasse erfordert die Einhaltung der durch die Model-Klasse festgelegten Groß- und Kleinschreibung.

Jeder Node-Instanz kann – falls vorhanden – eine Datei mit Vergleichsmessungen zugeordnet sein. Das Format des Dateinamens ist *node_xxxxxx_obs.asc* bzw. *node_xxxxxx_obs.npy*.

Sämtliche Eingangsreihen müssen in derselben zeitlichen Auflösung über denselben Zeitraum vorliegen. Eine frameworkinterne Ergänzung von Fehlwerten erfolgt nicht. Unabhängig vom Format enthalten die Dateien zugunsten der Einlesezeit lediglich die Eingangswerte selbst, aber keinen Zeitstempel. Um die Gefahr der Verwendung falscher Zeiträume einzuschränken, ist jedem Ordner eine Datei mit dem Namen *info.txt* beizulegen. Diese enthält – wie die Hauptsteuerdatei – die Einträge *timestep*, *date_startData* und *date_endData*. Bei Initialisierung des Frameworks wird auf Übereinstimmung geprüft.

Im ASCII-Format entspricht jede Zeile einem Datum. Benötigt eine Model-Klasse mehrere Werte für eine Eingangsgröße, beispielsweise um diese auf Untereinheiten aufzuteilen, sind diese innerhalb einer Zeile per Tabulator-Zeichen zu trennen.

Deutlich schnelleres Lesen und Schreiben ermöglicht das binär-Format des Moduls NumPy zur Speicherung von ein- oder mehrdimensionaler Arrays. Jeder *.npy*-Datei sind Informationen zur enthaltenen Datenstruktur beigefügt. Folgendes Beispiel veranschaulicht die Verwendung:

```
>>> import numpy as np
>>> Q1 = np.array(Q1)
>>> np.save('Q.npy', Q1)
>>> Q2 = np.load('Q.npy')
```

Das Modul NumPy wird geladen und ist im Folgenden über *np* anzusprechen. Anschließend

wird sichergestellt, dass der Vektor *Q1* als Array-Instanz vorliegt. Mit dem Befehl *save* erfolgt das Speichern (da nicht anders angegeben) im aktuellen Arbeitsverzeichnis, mit *load* das Einlesen. *Q1* und *Q2* sind bezüglich Struktur und Inhalt identisch.

1.10.2 Zeitreihen-Ausgabeformate (.asc, .npy, .bin)

Die Ausgabe von Zeitreihen durch die Routinen des Frameworks kann in drei Formaten erfolgen. Zum getrennten Speichern verschiedener Simulationsläufe dienen das ASCII- sowie das binäre NumPy-Format. Diese werden im Abschnitt 1.10.1 beschrieben. Das dritte Format ist binär und dient der Speicherung zahlreicher Simulationsergebnisse in einer Datei. Dieses Format wird zu einem späteren Zeitpunkt in die Dokumentation aufgenommen.

Unterschiede ergeben sich in den Dateinamen.

Je nach Typ der ausschreibenden Instanz entsprechenden die Namen der Eingangsdateien im Standardfall dem Format *element_xxxxxx_sim, node_xxxxxx_sim* oder *model_xxxxxx_X* zuzüglich der Endung *.asc*, *.npy* oder *.bin*. Um Ergebnisse verschiedener Simulationsläufe im selben Ordner hinterlegen zu können, kann den Dateinamen ein weiteres Suffix angehängt werden. Das Namensformat lautet dann *element_xxxxxx_sim_Y, node_xxxxxx_sim_Y* oder *model_xxxxxx_X_Y* zuzüglich der Dateiendung, wobei *Y* Platzhalter für einen vom Anwender übergebenen Text oder eine Ganzzahl ist.

Grundsätzlich ist zu beachten, dass Ausgabedateien von Speichergrößen gegenüber denen von Flussgrößen einen zusätzlichen Eintrag enthalten (im ASCII-Format eine zusätzliche Zeile). Speichergrößen werden für den Start- und den Endzeitpunkt der Simulation sowie alle dazwischen befindlichen Zeitpunkte ausgegeben. Flussgrößen dagegen für die zwischen Start- und Endzeitpunkt liegenden Intervalle.

1.11 Integrationsalgorithmen

Unter ad-hoc-Strategie soll die modellspezifische Lösung von Differenzialgleichungen verstanden werden, die im Quelltext der Modellgleichungen selbst implementiert ist, i. d. R. auf Genauigkeitsüberprüfungen verzichtet und oftmals von sequentiellem Charakter ist. Zustandsraum-Strategie bezeichnet hingegen die modellübergreifend standardisierte, getrennte Formulierung von Fluss- sowie Speichergrößenberechnungen, welche den Einsatz allgemeingültiger Algorithmen zur numerischen Integration gewöhnlicher Differenzialgleichungen ermöglicht.

Der Anwender kann, falls das eingesetzte Modell sowohl eine ad-hoc-Strategie und eine Zustandsraum-Strategie vorsieht, zwischen diesen wählen, indem er in die Modell-Initialisierungsdatei (Abschnitt 1.9.3) das Schlüsselwort *solver* gefolgt von *adhoc* oder *stateSpace* einträgt. Wird das Schlüsselwort nicht gesetzt, kommt die vom Modellentwickler vorgesehene Standardstrategie zum Einsatz. Zudem können vom Modellentwickler andere Optionen als *adhoc* und *stateSpace* zugelassen werden, beispielsweise um den Anwender einen bestimmten Integrationsalgorithmus auswählen zu lassen.

Im Folgenden wird der bislang einzige im Framework implementierten Integrationsalgorithmus beschrieben. Ein weiterer, impliziter Algorithmus und evtl. eine Kopplung von expliziten mit impliziten Verfahren ist geplant. Weitere Hinweise zur programmtechnischen Umsetzung der Integrationsalgorithmen finden sich im Abschnitt 1.13.2.

1.11.1 Explizites Verfahren

Die Funktion *adaptiveEulerRungeKutta* basiert auf dem in (Dahmen & Reusken, 2006) beschriebenen Algorithmus zur adaptiven Schrittweitensteuerung. Hierbei erfolgt die Justierung der Rechenschrittweite eines Einschrittverfahrens so, dass ein vorgegebener Integrationsfehler knapp unterschritten wird. So wird eine möglichst geringe Rechendauer bei Einhaltung einer gewünschten Genauigkeit angestrebt. Der Integrationsfehler wird mit Hilfe der Unterschiede geschätzt, die sich bei ganzer und halber Rechenschrittweite für die Speichergrößen ergeben. Wie im Folgenden beschrieben wurden zwei Funktionalitäten ergänzt.

In hydrologischen Modellen dürfen die Systemzustände, also die Speichergrößen, i. d. R. nur einen bestimmten Wertebereich annehmen. Derartige Restriktionen sind in ad-hoc-Lösungen von Differenzialgleichungen leicht zu implementieren, werden durch die gängigen numerischen Integrationsalgorithmen aber nicht berücksichtigt, wie ausführlich in Clark & Kavetski (2010) diskutiert. In *adaptiveEulerRungeKutta* werden daher nach der Aktualisierung der Speicherzustände Restriktionsverletzungen behoben und die maximale Über- oder Unterschreitung der zulässigen Grenzen als zusätzliches Maß des Integrationsfehlers verwendet. Diese scheinbar simple Vorgehensweise kann die Effizienz numerischer Integrationsalgorithmen je nach Häufigkeit der Verletzung von Restriktionen drastisch reduzieren, was z. B. die Parameterkalibrierung bestimmter Modelltypen erschwert. Hierauf ist bereits bei der Implementierung eines Modells zu achten und ggf. sollte auf die Verwendung von Grenzwerten in der Formulierung der Differenzialgleichungen verzichtet werden, wofür die in Abschnitt 1.13.3 erläuterten Methoden zur Verfügung stehen.

Das klassische explizite, 4-stufige Runge-Kutta-Verfahren bewältigt viele komplizierte Integrationsprobleme mit relativ großer Rechenschrittweite. Entsprechend dem von (Dahmen & Reusken, 2006) beschriebenen Algorithmus sind hiermit allerdings 12 Berechnungen aller Flussgrößen in jedem Rechenschritt notwendig (4-stufiges Verfahren einmal auf den ganzen und zweimal auf den halben Rechenschritt angewandt). Bei starken Systemreaktionen, z. B. bei hohen Niederschlagsintensitäten, ist dieser Aufwand gerechtfertigt. Bei trägem Systemverhalten, z. B. bei längeren Trockenwetterperioden, lässt sich dagegen auch mit dem einfachen expliziten Euler-Verfahren bei einer Rechenschrittweite gleich der Zeitschrittweite eine akzeptable Genauigkeit erzielen. Daher wird in *adaptiveEulerRungeKutta* zu Beginn eines Zeitschrittes, falls der vorherige Zeitschritt in nur einem Rechenschritt gelöst wurde, der Euler-Algorithmus angewandt. Sollte dieser die gewünschte Genauigkeit nicht in einem Rechenschritt erzielen, wird für den gesamten restlichen Zeitschritt auf das Runge-Kutta-Verfahren zurückgegriffen.

1.12 Anwendungsschritte

In diesem Abschnitt wird das Framework anhand seiner Anwendungsmöglichkeiten veranschaulicht. Die Installation von Python sowie das Vorhandensein der erforderlichen Verzeichnisstruktur samt Steuerdateien und Zeitreihen-Eingabedateien werden vorausgesetzt. Aktuelles Arbeitsverzeichnis soll die oberste Ebene der Verzeichnisstruktur, also der Ordner *HydPy* sein, in dem Python gestartet wird. Der Projektname lautet *Rhein_gesamt*.

1.12.1 Initialisierung

Zunächst ist die Klasse HydPy aus dem Modul HydPy zu importieren:

```
>>> from HydPy import HydPy
```

Alle weiteren Klassen werden – sobald erforderlich – automatisch importiert.

Um ein projektbezogenes Objekt der Klasse HydPy zu instanziieren ist an deren Konstruktor der Projektname als String zu übergeben:

```
>>> rhein = HydPy('Rhein _ gesamt')
```

Strings werden durch (halbe oder ganze) Anführungszeichen kenntlich gemacht. *rhein* dient im Folgenden als Referenz auf die HydPy-Instanz, welche sämtliche Arbeitsschritte sowie Datenzugriffe ermöglicht.

In Python können im Aufruf einer Funktion bzw. Methode grundsätzlich positions- und schlüsselwortbezogene Parameterübergaben erfolgen. Bei letzteren ist der Parametername mit Gleichheits-Zeichen voranzustellen:

```
>>> rhein = HydPy(nameProject='Rhein _ gesamt')
```

Dem HydPy-Konstruktor lassen sich verschiedene weitere, optionale Parameter übergeben. Wird hiervon kein Gebrauch gemacht, ist dies gleichbedeutend mit der Übergabe des jeweils definierten Standardwertes. (*nmbCPUs* wird erst in der Anleitung zur Steuerung des multiprocessing-Modus in Abschnitt 1.12.5 eingeführt).

printProgress ist mit *True* vorbelegt. Durch Belegung mit *False* wird das Ausschreiben des bislang erreichten Standes der Initialisierung unterdrückt:

```
>>> rhein = HWU('Rhein _ gesamt', printProgress=False)
```

printProgress ist lediglich für den Zeitraum der Initialisierung gültig.

Im Standardfall werden durch Aufruf des HydPy-Konstruktors sämtliche in den Steuerdateien definierten Model-Instanzen vollständig initialisiert. Sämtliche Eingabe-Zeitreihen werden in Arrays eingelesen und auch die für spätere Berechnungen erforderlichen Arrays vorab angelegt, was den Arbeitsspeicher überlasten kann. In diesem Fall ist der mit *False* vorbelegte Parameter *dynamicInit* auf *True* zu setzen:

```
>>> rhein = HWU('Rhein _ gesamt', dynamicInit=True)
```

So wird die Model-Instanziierung zunächst ausgelassen. Erst bei Aufruf der *run*-Methode der HydPy-Klasse erfolgt – für jede Element-Instanz getrennt – eine temporäre Initialisierung für die Simulationsdauer des Modells. Der Vorteil des geringeren Speicherbedarfes geht mit dem Nachteil der erhöhten Rechenzeit bei wiederholten Simulationen einher. Zudem ist unter Rückgriff auf die dynamische Initialisierung der interaktive Zugriff auf die Model-Instanzen vor oder nach der Simulation zwangsläufig unterbunden.

Möglichkeiten zur räumlichen Selektion bieten die Parameter *select*, *deselect* und *useOldUpstreamSim*. Erstere sind mit *None* vorbelegt, letzteres mit *False*. Die räumliche Selektion ist für die gesamte Lebensdauer einer HydPy-Instanz wirksam. Nur aktivierte Node- und Element-Instanzen werden in Simulationen einbezogen.

Über *select* werden die zu aktivierenden Node- und Element-Instanzen bekannt gegeben. Bei Übergabe von *None* werden sämtliche Instanzen selektiert. Ansonsten ist ein Tupel mit drei Einträgen gefordert:

```
>>> rhein = HWU('Rhein_gesamt', select=('neckar2',[],True))
```

Der erste Tupeleintrag benennt Node-, der zweite Element-Instanzen. Der dritte Tupeleintrag legt fest, ob alle oberhalb gelegenen Node- und Element-Instanzen automatisch in die Selektion aufzunehmen sind. Im Beispiel wird die Node-Instanz mit den Namen *neckar2* direkt ausgewählt. Der dritte Eintrag ist *True*, womit die Aufnahme sämtlicher oberhalb gelegener Node- und Element-Instanzen in die Auswahl erfolgt. Beim Eintrag *False* wäre das nicht der Fall. Element-Instanzen werden im Beispiel nicht direkt ausgewählt, was durch die leere Liste *[]* angezeigt wird.

Mittels *select* lassen sich leicht ganze Flussgebiete oberhalb eines Bezugspunktes aktivieren. Die Auslassung einzelner darin enthaltener Gebiete (z. B. bereits kalibrierte Kopfgebiete) mittels *select* ist zumeist aufwendig: Der dritte Tupeleintrag wäre mit *False* zu belegen, womit die ersten beiden Tupeleinträge vollständige Listen aller zu aktivierenden Node- und Element-Instanzen erforderten. Eleganter ist der Ausschluss der Kopfgebiete über *deselect*:

```
>>> rhein = HWU('Rhein_gesamt', select=('neckar2',[],True),
                deselect=('neckar1',[],True))
```

Alle Node- und Element-Instanzen oberhalb von und einschließlich *neckar2* werden vorläufig selektiert. Anschließend erfolgt der Ausschluss von *neckar1* sowie der oberhalb hiervon gelegenen Instanzen aus der Selektion. Letztendlich aktiviert werden nur die zwischen *neckar1* und *neckar2* liegenden Instanzen, ausgeschlossen *neckar1* und eingeschlossen *neckar2*.

Dem simulierten Abfluss für *neckar2* fehlt bei der so festgelegten räumlichen Selektion der Anteil von *neckar1*. Hierfür steht der Schlüsselwortparameter *useOldUpstreamSim* zur Verfügung. Durch Belegung mit *True* lesen sämtliche inaktiven Node-Instanzen, die direkt oberhalb aktiver Element-Instanzen liegen, ihre zuvor simulierte Ausgabe ein und reichen diese bei folgenden Simulationen weiter:

```
>>> rhein = HWU('Rhein_gesamt', select=('neckar2',[],True),
                deselect=('neckar1',[],True), useOldUpstreamSim=True)
```

Im Beispiel wird vorausgesetzt, dass die Ausgabegröße der Node-Instanz *neckar1* zuvor simuliert und ausgeschrieben wurde. Falls die ausgeschriebene Datei ein zusätzliches Suffix im Namen enthält (siehe Abschnitt 1.10.2), ist dieses als String anstelle von *True* an *useOldUpstreamSim* zu übergeben.

Den ersten beiden Einträgen der Tupel für *select* und *deselect* sind einzelne oder in Listen zusammengefasste Namen oder IDs (laufender Index, siehe Abschnitt 1.9.2) von Node- bzw. Element-Instanzen zu übergeben. Bei Belegung von *select* und *deselect* mit *None*, ist diejenige von *useOldUpstreamSim* irrelevant.

Auch nicht aktivierte Node- und Element-Instanzen werden initialisiert, sodass interaktiv auf diese zuzugreifen ist. Lediglich die viel Arbeitsspeicher benötigenden Modell- und Array-Instanzen werden bei der Initialisierung ausgespart.

Räumliche Selektion und dynamische Initialisierung schließen sich nicht aus.

1.12.2 Interaktiver Zugriff

Nach der nicht dynamischen Initialisierung besteht die direkte Zugriffsmöglichkeit auf sämtliche Member aller instanziierten Klassen des Frameworks. So lässt sich der Umgang mit den Methoden des Frameworks direkt erproben und die in den jeweiligen Model-Instanzen abgespeicherten Daten mit denen der Steuer- bzw. Zeitreihen-Dateien vergleichen oder auch modifizieren. Die Anzahl der Element-Instanzen ist beispielsweise abzufragen durch:

```
>>> rhein.nmbElements
```

Es ist wichtig zu beachten, dass Attribute im Framework nicht geschützt werden. Beispielsweise ließe sich *nmbElements* mit einem anderen Wert beliebigen Typs belegen. Dieser erhöhte Freiheitsgrad für den Anwender geht mit einem erhöhten Risiko für schwer nachvollziehbare Fehler einher. Es wird empfohlen, nach Möglichkeit auf implementierte Methoden zur Änderung von Attributen zurückzugreifen.

Die Element- und Node-Instanzen selbst sind in jeweils ein Objekt vom Typ List eingehängt, das selbst Attribut der HydPy-Instanz ist. Da Indices in Python von Null beginnend gezählt werden, lässt sich beispielsweise auf die erste Element-Instanz zugreifen über:

```
>>> rhein.elements[0]
```

Die Element-Instanz übergibt beispielsweise die Ausgabe ihrer Model-Instanz an die Variable *Q* über den Befehl:

```
>>> Q = rhein.elements[0].getOutput()
```

Hierbei ist zu beachten, dass in Python Übergaben i. d. R. per Referenz erfolgen. Die Element-Instanz reicht im Beispiel eine Referenz von der Model-Instanz zur Variable *Q* weiter, nicht die Werte als solche. Das verringert den Bedarf an Arbeitsspeicher und Rechenzeit. Dadurch referenzieren *Q* und das entsprechende Attribut der Model-Instanz aber dieselbe Array-Instanz, sofern dieses durch die Programmierung der Model-Klasse nicht explizit ausgeschlossen ist. Daher ist eine Änderung „in“ *Q* ggf. dasselbe wie eine Änderung „in“ der Model-Instanz. Im Zweifelsfall sollte ein solches Verhalten durch vorheriges Anlegen einer Kopie unterbunden

werden, z. B. durch die *copy*-Methode der Array-Klasse:

```
>>> Q = Q.copy()
```

Der Zugriff auf die Model-Instanz erfolgt über:

```
>>> rhein.elements[0].model
```

Welche Member *model* enthält, hängt zum Teil vom Modelltyp ab. Jede Model-Klasse sollte aber eine Instanz der ParameterHandler-Klasse unter der Bezeichnung *parameter* enthalten. Der Wert des Modellparameters *K* ließe sich über folgenden Befehl abfragen:

```
>>> rhein.elements[0].model.parameter.K
```

Zudem sollte jede Model-Klasse von der ArrayHandler-Klasse Gebrauch machen. Deren Instanz sind die Sub-Instanzen *input*, *flux* und *storage* angehängt. Auf eine Eingangs-Temperaturreihe *T* ließe sich zugreifen über:

```
>>> rhein.elements[0].model.arrays.input.T
```

Für die Aufteilung der Fluss- und Speichergrößen auf die FluxOrStorageHandler-Instanzen *flux* und *storage* liegen mehrere Gründe wie beispielsweise die unterschiedliche Zeitreihenlänge vor (siehe Abschnitt 1.10.2). In *storage*-Arrays hinterlegte Werte beziehen sich auf Zustandsgrößen zu definierten Zeitpunkten. Ein Beispiel wäre der Bodenwassergehalt. In *flux-Arrays* hinterlegte Werte beziehen sich hingegen auf Intervalle. Damit umfasst *flux* echte Flüsse wie den Niederschlag, aber auch Reihen von Intervall-Mittelwerten u. Ä. von Zustandsgrößen wie beispielsweise die Tagesmitteltemperatur.

Damit ist die Objektstruktur des Frameworks im singleprocessing-Modus grob umrissen. Es ist zu beachten, dass die im Folgenden beschriebenen Anwendungsmöglichkeiten zum Teil auf verschiedenen Ebenen durchgeführt werden können. Die Methode der höchsten Ebene ist dabei die sicherste und komfortabelste Wahl. Nur wenn deren Freiheitsgrade nicht ausreichen, sollte auf die Methoden der niedrigeren Ebenen zurückgegriffen werden.

1.12.3 Simulation

Nach der Initialisierung lässt sich die Simulation über die *run*-Methode starten:

```
>>> rhein.run()
```

Im singleprocessing-Modus werden die Element- und Node-Instanzen in geeigneter Reihenfolge ausgewählt, ggf. die Simulation der eingebundenen Model-Instanz durchgeführt und abschließend die Ausgabegrößen weitergereicht. Der Ablauf im multiprocessing-Modus wird im Abschnitt 1.12.5 ausführlicher behandelt.

Der optionale Parameter *printProgress* ist mit *True* vorbelegt. Es gilt dasselbe wie für den gleichnamigen Parameter im Aufruf des HydPy-Konstruktors (Abschnitt 1.12.1).

Der optionale Parameter *resetArrays* ist ebenfalls mit *True* vorbelegt. Damit werden standardmäßig vor der Simulation sämtliche unter *flux* oder *storage* gespeicherten Arrays auf null bzw. ihre Anfangswerte zurückgesetzt. Zur Erhöhung der Rechengeschwindigkeit kann dies durch

Übergabe des Wertes *False* verhindert werden. Sinnvoll ist das aber nur, wenn alle im Projekt eingesetzten Model-Klassen ihre Arrays implizit oder explizit zurücksetzen bzw. ein Zurücksetzen nicht benötigen. Im Zweifelsfall sind durch wiederholte Simulationen mit übergebenem *False*-Wert erzielte Ergebnisse auf Übereinstimmung zu prüfen.

1.12.4 Ergebnisausgabe

Zur Ergebnisausgabe stellt die HydPy-Klasse drei Methoden bereit: *writeArrays* schreibt simulierte Zeitreihen, *writeContrPars* und *writeInitVals* Steuerdateien.

In welches Verzeichnis, mit welcher Benennung und in welchem Format die Methode *writeArrays* Zeitreihen ausschreibt, ist in den Abschnitten 1.9.1 und 1.10.2 erläutert. In diesem Abschnitt wird die Verwendung der Methode hauptsächlich in Hinblick auf die gezielte Speicherung ausgewählter Zeitreihen beschrieben.

Die Methode wird wie folgt aufgerufen:

```
>>> rhein.writeArrays()
```

Es ist zu beachten, dass durch diesen Funktionsaufruf – trotz korrekter Initialisierung und ggf. Simulation – keinerlei Zeitreihen ausgeschrieben werden. Das liegt an den Standardwerten der optionalen Parameter, die das unbeabsichtigte Schreiben großer Datenmengen verhindern.

Die Parameter *select* und *deselect* sind analog zu den Parametern des HydPy-Konstruktors einzusetzen (siehe Abschnitt 1.12.1). Wie bei der HydPy-Initialisierung sind beide mit *None* vorbelegt. Ein *None*-Wert sowohl für *select* als auch *deselect* bedeutet für das Ausschreiben von Zeitreihen, dass sämtliche Node- und Element-Instanzen aktiviert werden. Bei der HydPy-Initialisierung deaktivierte Node- und Element-Instanzen bleiben beim Schreiben der Zeitreihen grundsätzlich inaktiv – unabhängig von den an *writeArrays* übergebenen Selektionseinstellungen.

Über den Parameter *outputArrays* ist festzulegen, ob aktivierte Node- und Element-Instanzen ihren „output“ in Dateien schreiben sollen. Die Standardbelegung ist *False*. Durch Setzen von *True* schreiben sowohl Node- als auch Element-Instanzen. Alternativ kann ein Tupel mit zwei booleschen Werten übergeben werden: der erste Wert spricht die Node, der zweite die Element-Instanzen an. In folgendem Beispiel schreiben lediglich die aktivierten Node-Instanzen aus:

```
>>> rhein.writeArrays(outputArrays=(True,False))
```

Über den Parameter *modelArrays* wird eine Auflistung aller internen Fluss- und Speichergrößen bekannt gegeben, die von den aktivierten Model-Instanzen auszugeben sind. Die Standardbelegung ist *None*. Diese führt, genauso wie die Belegung mit leeren Listen oder Dictionaries, zu keinerlei Modellausgaben. Die Steuerung der Modellausgabe über eine Liste funktioniert wie folgt:

```
>>> rhein.writeArrays(modelArrays=['Q1','Q2'])
```

Die Liste enthält modellinterne Bezeichnungen von Fluss- oder Speichergrößen als Strings. Jede aktivierte Model-Instanz schreibt – falls vorhanden – die durch die entsprechenden Variablen referenzierten Arrays aus. Ist eine Variable nicht definiert, erfolgt keine Rückmeldung an den

Anwender. Da i. d. R. mehrere Model-Klassen mit verschiedenen Benennungen von Fluss- und Speichergrößen in einem Projekt zum Einsatz kommen, ist die Übergabe einer unbekannten Bezeichnung an eine Model-Instanz zumeist nicht als Fehler anzusehen.

Etwas aufwendiger, dafür aber sicherer, ist die Übergabe eines Dictionaries:

```
>>> rhein.writeArrays(modelArrays={'model1' : ['Q1','Q2'], 'model2' : ['Q']})
```

Im Beispiel wird jede aktive Instanz der Model-Klasse *model1* zum Speichern der Flussgrößen *Q1* und *Q2*, sowie jede aktive Instanz der Model-Klasse *model2* zum Speichern der Flussgröße *Q* aufgefordert. Aufgrund der konkreten Zuordnung zwischen Model-Klasse und Fluss- oder Speichergröße wird die Übergabe einer nicht in der Model-Klasse definierten Benennung als Fehler aufgefasst und gemeldet. Ist eine instanziierte Model-Klasse nicht im Dictionary enthalten, entspricht dieses der Übergabe einer leeren Liste für die Model-Klasse und führt zu keiner Rückmeldung.

In Abschnitt 1.10.2 wird die Möglichkeit beschrieben, Namen von Ausgabedateien durch ein Suffix zu ergänzen. So lassen sich die Ergebnisse verschiedener Simulationsläufe in einem Ordner speichern. Hierfür ist dem Parameter *suffix* ein String oder ein Integer zu übergeben. Ein Integer-Wert wird automatisch in einen wenigstens sechsstelligen String umgewandelt, der falls erforderlich linksseitig mit Nullen aufgefüllt wird. Die folgenden beiden Befehle sind äquivalent:

```
>>> rhein.writeArrays(outputArrays=True, suffix=12)
>>> rhein.writeArrays(outputArrays=True, suffix='000012')
```

Durch Belegung des Parameters *mode* mit dem String *asc*, *npy* oder *bin* kann das für den aktuellen Schreibvorgang gültige Ausgabeformat abweichend zur Angabe in der Hauptsteuerdatei gewählt werden:

```
>>> rhein.writeArrays(outputArrays=True, mode='npy')
```

Standardmäßig ist *mode* mit *None* belegt, womit die Angabe in der Hauptsteuerdatei zum Tragen kommt.

Der Schlüsselwortparameter *skipFirst* gibt an, wie viele Zeitschritte zu Simulationsbeginn nicht auszuschreiben sind. Zu übergeben ist ein Integer-Wert, die Standardbelegung ist Null. Hintergrund ist die effiziente Datenspeicherung in Fällen, bei denen die ersten berechneten Werte als „warm up“ zu verwerfen sind. Von dieser Option sollte nur bei ausreichender Dokumentation Gebrauch gemacht werden, um die Nachvollziehbarkeit der Ergebnisse nicht zu gefährden. Im folgenden Beispiel werden die ersten 365 Werte verworfen:

```
>>> rhein.writeArrays(outputArrays=True, skipFirst=365)
```

Durch Aufruf der HydPy-Methode *writeContrPars* werden Modell-Initialisierungsdateien gespeichert. So lassen sich beispielsweise durch Kalibrierung ermittelte Modellparameter HydPy-kompatibel hinterlegen. Die beiden optionalen Parameter sind *folder* und *overwrite*. Ersterer bestimmt den Zielordner der Dateien. Standardbelegung ist *None*, wodurch der Projektname als Ordnername fungiert. Der angegebene Ordner wird neu angelegt. Falls bereits ein gleichnami-

ger Ordner vorhanden ist, kommt die Belegung von overwrite zum Tragen: die Standardbelegung *False* führt zu einer Fehlermeldung, *True* hingegen zum Überschreiben ohne Rückmeldung. Im folgenden Beispiel wird der Ordner *neuerOrdner* angelegt bzw. überschrieben, falls er bereits existiert:

```
>>> rhein.writeContrPars(folder='neuerOrdner', overwrite=True)
```

Ähnliches wie für *writeContrPars* gilt für die Methode *writeInitVals*. Diese schreibt Anfangswertdateien aus. Hierbei kommt der zeitliche Aspekt hinzu: Über den Parameter *date* wird der Zeitpunkt festgelegt, für den die Systemzustände auszuschreiben sind. Die Standardbelegung ist *None* und führt zur automatischen Auswahl des Endzeitpunktes der Simulation. Für *date* kann ein String entsprechend den in Abschnitt 1.9.1 genannten Konventionen oder direkt ein datetime-Objekt übergeben werden. Wird für den standardmäßig mit *None* belegten Schlüsselwertparameter *folder* kein String übergeben, wird der Name des neuen Ordners aus dem Datum der Systemzustände gebildet. Folgende Befehle sind äquivalent:

```
>>> rhein.writeInitVals(date='2012-12-24_16:15', folder='2012-12-24_16:15')
>>> rhein.writeInitVals(date=datetime(2012,12,24,16,15'), overwrite=False)
```

1.12.5 Multiprocessing

Wie im Abschnitt 1.7 beschrieben, weicht die Handhabung des Frameworks im multiprocessing-Modus bei den üblichen Arbeitsschritten kaum von der im singleprocessing-Modus ab. Der multiprocessing-Modus wird durch Übergabe eines Integer-Wertes größer eins an den Schlüsselwortparameter *nmbCPUs* aktiviert:

```
>>> rhein = HWU('Rhein _ gesamt', nmbCPUs=4)
```

nmbCPUs begrenzt die maximal bei der Simulation in Anspruch genommene Rechenleistung. Wird der obige Befehl auf einem Rechner mit 8 Kernen ausgeführt, liegt die maximale Prozessorauslastung bei ca. 50 %. Für die Dauer einer Simulation lässt sich diese über den gleichnamigen Parameter der *run*-Methode ändern:

```
>>> rhein.run(nmbCPUs=8)
```

Im diesem Beispiel beträgt die maximale Prozessorauslastung 100 %. Dieser Maximalwert wird nicht während der gesamten Simulationsdauer erreicht. Daher können von der Anzahl der verfügbaren Rechenkerne abweichende Belegungen für *nmbCPUs* zum schnelleren Abschluss der Gesamtsimulation führen, was für den Einzelfall zu prüfen ist.

Die Hauptursache für die zeitweise Unterschreitung der insgesamt zur Verfügung stehenden Prozessorleistung ist, dass die Simulation nicht vollständig parallelisierbar ist. Flussab gelegene Model-Instanzen können erst rechnen, wenn oberhalb gelegene Instanzen ihre Ausgabe ermittelt und weitergeleitet haben. Das Zusammenlaufen eines Flussnetzes führt somit zu einer relativ geringen Auslastung der Rechenkerne gegen Simulationsende. Eine weitere Verschärfung dieses Problems kann sich durch die ungünstige Vernetzung von Model-Instanzen mit unterschiedlicher Rechendauer ergeben.

Ist die durchschnittliche Prozessorauslastung bei einem bestehenden Projekt nicht befriedigend, empfiehlt sich als erstes die Neubestimmung des HydPy-Attributes *runningOrder*. Hierbei handelt es sich um eine Liste, die bei der HydPy-Initialisierung durch die Methode *determineRunningOrder* ermittelt wird:

```
>>> rhein.runningOrder = rhein.determineRunningOrder()
```

runningOrder legt im singleprocessing-Modus eindeutig fest, in welcher Reihenfolge Node- und Element-Instanzen auszuführen sind. Dazu enthält sie in der Simulationsreihenfolge geordnete Tupel, welche jeweils eine Instanz über deren ID sowie einen der Strings *node* oder *element* bezeichnen.

Im multiprocessing-Modus gibt *runningOrder* keine feste, sondern eine zu favorisierende Simulationsreihenfolge vor: Über Threading werden sämtliche ElementServer- und NodeServer-Instanzen quasi gleichzeitig gestartet. Unmittelbar anschließend werden diese ggf. in einen Schlafmodus versetzt, bis über eine Queue alle erforderlichen Oberlieger-Zeitreihen eingegangen sind. Die nicht laufzeitkritischen NodeServer-Instanzen reichen den kumulierten Input dann unmittelbar weiter. Für die laufzeiteffiziente Abarbeitung der Model-Instanzen dagegen werden durch die Klasse ElementThread weitere Locking-Mechanismen eingeführt. ElementThread-Instanzen werden für jeden Simulationslauf neu initialisiert, weshalb sie in Abbildung 1.3 nur lose an die HydPy-Klasse und die ElementClient-Klasse angebunden dargestellt sind. Durch diese wird erstens über ein Condition-Lock sichergestellt, dass ein Modell mit vollständigem Input erst rechnet, wenn kein anderes Modell mit vollständigem Input und laut *runningOrder* höherer Priorität wartet. Und zweitens über ein Semaphore-Lock, dass die Anzahl der parallel rechnenden Modelle die Anzahl der vom Benutzer vorgegebenen Rechenkerne nicht übersteigt.

determineRunningOrder schätzt die Priorität der Node- und Element-Instanzen – sprich ihre Reihenfolge in *runningOrder* – umso höher ein, je mehr Node- und Element-Instanzen flussab folgen. Bei stark unterschiedlichen Rechenzeiten der in einem Projekt implementierten Model-Instanzen können durch alternative Festlegungen der Priorität nennenswerte Steigerungen der durchschnittlichen Prozessorauslastung zu erzielen sein. Beispielsweise könnte statt der Anzahl der flussab gelegenen Instanzen deren geschätzte oder gemessene Rechendauer herangezogen werden.

Der multiprocessing-Modus basiert auf der Initialisierung und Ausführung von Element- und Node-Instanzen innerhalb eigenständiger Prozesse, wobei die Instanzen entsprechend dem Client-Server-Modell miteinander sowie mit der HydPy-Instanz in Verbindung stehen. Entsprechend ist der multiprocessing-Modus mit der Verteilung auf Rechenclustern prinzipiell kompatibel. Was fehlt, ist eine Routine zur günstigen Verteilung der dauerhaft initialisierten Node- und Element-Server. Zur gleichmäßigen Auslastung eines Clusters ist *runningOrder* zu berücksichtigen oder alternativ zu ermitteln. Als günstiger Einstiegspunkt bietet sich dem versierteren Anwender die HydPy-Methode *_initServersMultiCPUs* an.

Das Schließen der eigenständigen Prozesse der Element- und Node-Instanzen erfolgt nicht automatisch durch den Python-Interpreter. Wird der Hauptprozess der HydPy-Instanz nach dem Aufruf der zusätzlichen Prozesse ohne Vorkehrungen geschlossen, bleiben diese zeitlich unbe-

grenzt aktiv. Vom Anwender sollte daher z. B. vor einer HydPy-Neuinitialisierung sicherheitshalber der folgende Befehle ausgeführt werden:

```
>>> rhein.terminate()
```

Im multiprocessing-Modus beendet *terminate* alle zuvor gestarteten zusätzlichen Prozesse und löscht die entsprechenden Clients. Im kritischen Fehlerfall versucht die HydPy-Instanz *terminate* automatisch auszuführen.

Als Rückfalloption dient die HydPy-Methode *taskkill*:

```
>>> rhein.taskkill()
```

taskkill bietet verschiedene Möglichkeiten, Python-Prozesse zu schließen. Bei Belegung des einzigen Parameters *mode* mit dem Standardwert *1* wird versucht, sämtliche durch HydPy-Instanzen gestartete Python-Prozesse zu schließen. Dieses umfasst auch Prozesse von bereits deinitialisierten HydPy-Instanzen. Bei Übergabe des Wertes *2* werden sämtliche Python-Prozesse geschlossen, die nach dem Konstruktor-Aufruf der aktuellen HydPy-Instanz gestartet wurden. Die Belegung von *mode* mit *3* schließt sämtliche Python-Prozesse, ausschließlich dem ausrufenden. Unter Windows schließt *taskkill* bei Übergabe des Wertes *4* mit Hilfe des Befehlszeilenprogramms Taskkill sämtliche aktiven Python-Prozesse. Der von *taskkill* aufgerufene Kommandozeilenbefehl lautet:

```
>>> taskkill.exe /F /IM pythonw.exe
```

Im Cluster ist dieser Kommandozeilenbefehl auf jedem einzelnen Rechner auszuführen bzw. die HydPy-Methode *taskkill* entsprechend zu erweitern. *terminate* hingegen ist auch im Cluster unmittelbar funktionsfähig.

1.12.6 Assistenzfunktionen

Die HydPy-Klasse enthält verschiedene Methoden, die nur dem vereinfachten Umgang mit dem Framework dienen, von denen im Folgenden die wichtigsten beschrieben werden.

Mit *readObs* werden sämtliche aktiven Node-Instanzen zum Einlesen gemessener Zeitreihen aufgefordert (siehe Abschnitt 1.10.1). Der einzige Parameter *printIfMissing* ist mit *False* vorbelegt. Nur bei Übergabe von *True* werden dem Anwender die Namen fehlender Dateien gemeldet:

```
>>> rhein.readObs(printIfMissing=True)
```

Sollen nur einzelne Node-Instanzen einlesen, ist direkt auf deren gleichnamige Methode zurückzugreifen:

```
>>> rhein.nodes[0].readObs(printIfMissing=True)
```

Mit der Methode *name2ID* lassen sich die IDs bzw. laufenden Indices von Element- oder Node-Instanzen ermittelt. An erster Position im Aufruf ist ein entsprechender String oder eine Liste von Strings zu übergeben, an zweiter der Instanztyp *element* oder *node* als String:

```
>>> rhein.name2ID(['neckar1','neckar2'], 'node')
```

Zurückgegeben wird grundsätzlich eine Liste mit Ganzzahlen von Typ Integer.

Der Zugriff auf den Wert einer Zeitreihe bzw. eines Arrays zu einem bestimmten Zeitpunkt erfolgt im Framework grundsätzlich über einen laufenden Index. Dieser beginnt zum Startzeitpunkt der Simulation bei null. Die Methode *date2index* gibt den Indexwert für ein anzugebendes Datum zurück. Dieses Datum ist an erster Stelle in Funktionsaufruf als datetime-Objekt oder String zu übergeben. In letzterem Fall ist das im Abschnitt 1.9.1 definierte Format zu wählen oder über den optionalen Parameter *dateFormat* ein alternatives Format anzugeben. Folgende Aufrufe sind äquivalent:

```
>>> rhein.date2index(datetime(2012,12,24,16,15))
>>> rhein.date2index('2012-12-24_16:15')
>>> rhein.date2index('24.12.2012 16:15', dateFormat='d.m.Y H:M')
```

Es ist zu beachten, dass *date2index* zwei Werte ausgibt: der zweite bezieht sich auf den „echten“ Zeitpunkt einer Speichergröße, der erste dagegen auf das Intervall einer Flussgröße unmittelbar vor diesem Zeitpunkt. Folgender Befehl gibt damit zwangsläufig das Tupel *(-1, 0)* aus (siehe Abschnitt 1.10.2):

```
>>> rhein.date2index(rhein.pubDic['date_startSim'])
```

Das Gegenstück zu *date2index* ist die Methode *index2date*. An erster Stelle ist ein Index-Wert zu übergeben. Der optionale Parameter *asText* ist mit *False* vorbelegt, womit standardmäßig datetime-Objekte zurückgegeben werden. Bei Belegung von *asText* mit *True* werden die Daten als String im Standardformat ausgegeben, sofern über *dateFormat* kein anderes Format definiert wird. Aus dem für *date2index* ausgeführten Grund wird je ein Datum für Fluss- und Speichergrößen mit der Differenz einer Simulations-Zeitschrittweite ausgegeben.

Die Methode *getAreaUpstream* ermittelt die Gesamtfläche aller oberhalb liegenden Modell-Instanzen:

```
>>> rhein.getAreaUpstream(idOrName='neckar2', typ='node', printWarning=True)
```

Der Bezugspunkt wird über den Parameter *idOrName* definiert, welcher die ID oder den Namen einer Node- oder Element-Instanz erwartet. An den Parameter *typ* ist je nach Fall der String *node* oder *element* zu übergeben. Dient eine Element-Instanz als Bezugspunkt, wird deren Modell-Instanz-Fläche in die Summierung einbezogen. Bei der Standardbelegung des dritten Parameters *printWarning* mit *True* werden sämtliche Element-Instanzen aufgeführt, für die keine Teilfläche zu ermitteln ist. In solchen Fällen wird die Gebietsgröße eventuell unterschätzt. Allerdings ist zu bedenken, dass einer Modell-Instanz, welche z. B. einen Flussabschnitt repräsen-

tiert, oftmals keine Fläche zugeordnet ist. Bei ausschließlicher Verwendung von *area* zur Angabe der Teilgebietsgröße in den Modell-Initialisierungsdateien sollte die berechnete Gebietsgrößer daher auch bei Ausgabe von Warnungen korrekt sein.

1.13 Modell-Implementierung

Zur einfachen und einheitlichen Implementierung neuer Modell-Typen dient die Basisklasse Model. Der Anwender muss sich i. d. R. nur in Teile dieser Klasse einarbeiten, um eigene Modelle erstellen zu können. Durch zwei Befehlszeilen lässt sich ein neues Modell definieren, dass sämtliche Member der Basisklasse erbt:

```
>>> class newModel(Model):
>>>     pass
```

Die so abgeleitete neue Modellklasse verfügt beinahe über den vollständigen von HydPy geforderten Funktionsumfang. In diesem Abschnitt wird neben der Funktionsbeschreibung einzelner Methoden ausgeführt, ob und ggf. wie diese zu überlagern oder zu ergänzen sind. Eine Methode von Model wird beibehalten, wenn sie in der abgeleiteten Klasse nicht neu definiert wird. Somit steht sie anschließend unverändert zur Verfügung. Überlagerung findet dagegen durch Neudefinition in der abgeleiteten Klasse statt. Soll eine Methode nicht ersetzt sondern lediglich erweitert werden, muss eine Neudefinition in der abgeleiteten Klasse erfolgen, in der die Methode der Basisklasse aufgerufen wird. Zusätzliche Definitionen modellspezifischer Methoden können eingeführt werden, beispielsweise um die standardisierten Methoden der Basisklasse nicht zu überfrachten.

Im folgenden Beispiel behält *Rechner2* die Initialisierungmethode seiner Basisklasse *Rechner1* bei, d. h. mit Instanziierung des Objektes *rechner2* wird dessen Attribut *Ergebnis* mit dem Wert Null belegt. Die Methode *addiere* wird ergänzt. Ein Aufruf führt zunächst zur Addition und dann zur Ausgabe des Ergebnisses. Die Methode *subtrahiere* wird dagegen überschrieben, womit keine Subtraktion entsprechend *Rechner1*, sondern lediglich eine Warnmeldung erfolgt. Die Methode *multipliziere* wird zusätzlich definiert.

```
>>> class Rechner1(object):
>>>     def __init__(self):
>>>         self.Ergebnis = 0
>>>     def addiere(self, Wert):
>>>         self.Ergebnis += Wert
>>>     def subtrahiere(self, Wert):
>>>         self.Ergebnis -= Wert
>>>
>>> class Rechner2(Rechner1):
>>>     def addiere(self, Wert):
>>>         Rechner1.addiere(self, Wert):
>>>         print self.Ergebnis
>>>     def subtrahiere(self, Wert):
>>>         print "Rechner2 subtrahiert nicht"
>>>     def multipliziere(self, Wert):
>>>         self.Ergebnis *= Wert
>>>         print self.Ergebnis
>>>
>>> rechner2 = Rechner2()
```

1.13.1 Allgemeine Funktionalitäten

Die meisten der im Folgenden beschriebenen Methoden für die Handhabung der Zeitreihen-Arrays bieten die Möglichkeit, lediglich auf Teilzeiträume abzuzielen, was in den einzelnen Funktionsbeschreibungen nicht gesondert Erwähnung findet.

1.13.1.1 __init__

Der Konstruktur *__init__* erhält die Element-ID sowie allgemeine und modellspezifische Informationen in Form von Dictionaries von der aufrufenden Element-Instanz. Modellübergreifende Informationen wie die Länge der Zeitreihen-Arrays werden als Attribute gespeichert. Anschließend werden Umrechnungsfaktoren für Einheiten ermittelt und die Initialisierung der Parameter- und Array-Handler gestartet. *__init__* ist grundsätzlich zu ergänzen.

Die standardmäßige Zeiteinheit aller Parameter mit Zeitbezug ist ein Tag. Sind dagegen beispielsweise für die benutzerdefinierten Parameterwerte der Modell-Initialisierungsdateien (Abschnitt 1.9.3) sechs Stunden und für die modellspezifischen Standardwerte (Abschnitt 1.13.1.6) 30 Tage gewünscht, wären folgenden folgende Befehle hinzuzufügen:

```
>>> self.unitsTimestepUser = '6h'
>>> self.unitsTimestepStandard = '30d'
```

Zwingend erforderlich sind die Wahl der programmatischen Umsetzung *python*, *fortran* oder *cython* sowie der Strategie zur Lösung der Differenzialgleichungen *adhoc* oder *stateSpace* (Ab-

schnitt 1.13.2). Hierfür sollten Standardwerte gesetzt werden, die durch Verwendung der Schlüsselwörter *language* sowie *solver* in der Modell-Initialisierungsdatei vom Benutzer überschrieben werden können (siehe Abschnitt 1.9.3), beispielsweise:

```
>>> self.language = info.get('language', 'python')
>>> self.solver = info.get('solver', 'classic')
```

Zudem sind Import und Referenzierung der mittels Fortran oder Cython definierten Module zu regeln, beispielsweise:

```
>>> if self.language == 'fortran':
>>>     from fHBV96_zone import fhbv96_zone
>>>     self.fortran = fhbv96_zone
>>> else:
>>>     self.fortran = None
```

Der Aufruf des Konstruktors der Model-Basisklasse muss nach den genannten Änderungen und Ergänzungen erfolgen:

```
>>> Model.__init__(self, ID, pubDic, info)
```

1.13.1.2 initParameters

Der Konstruktor *__init__* ruft die Funktion *initParameters* unter Weitergabe der modellspezifischen Informationen auf. *initParameters* veranlasst die Ermittlung der primären, ggf. benutzerdefinierten Modellparameter und ruft – falls erforderlich – Methoden zur Parameterermittlung für numerische Integrationsalgorithmen (*getSolverPars*) sowie zur Berechnung sekundärer statischer (*calcStaticSecPars*) und dynamischer (*calcDynamicSecPars*) Parameter auf.

initParameters wird grundsätzlich ergänzt. Vor dem Aufruf der gleichnamigen Methode der Basisklasse ist das Dictionary *standardParameters* zu definieren. Dessen Schlüssel sind die Namen der primären Modellparameter als *string*. Die zugeordneten Objekte sind Tupel mit sechs Einträgen:

1. Standardwert, falls kein benutzerdefinierter Wert in der Modell-Initialisierungsdatei hinterlegt ist. Soll kein Standardwert vorgegeben werden, was den Benutzer zu einer entsprechenden Angabe zwingt, ist der Wert auf *nan* (not a number) zu setzen.
2. Unterer Grenzwert. Wird vom Benutzer ein kleinerer Parameterwert angesetzt, erfolgt eine Fehlermeldung. Dies kann durch Belegung des unteren Grenzwertes mit –inf (infinity) umgangen werden.
3. Oberer Grenzwert. Analog zum unteren Grenzwert.
4. Art der Anwendung von *unitsTimestepUser* bzw. *unitsTimestepStandard*, (Abschnitt 1.13.1.1). Mögliche Belegungen sind:
    a. 0: keine Modifikation des Parameterwertes, da dieser keinen Zeitbezug hat (z. B. Feldkapazität [Höhe]).
    b. 1: Modifikation eines Parameterwertes, der zur Bezugszeit proportional ist (z. B. maximale Infiltrationsrate [Höhe/Zeit])
    c. -1: Modifikation eines Parameterwertes, der zur Bezugszeit invers proportional ist (z. B. Schwerpunktlaufzeit [Zeit])
5. Dimensionalität. Mögliche Belegungen sind:
    a. 0: Der Parameter liegt skalar vor
    b. 1: Parameterwerte in einem eindimensionalen Array, z. B. um die Höhen verschiedener Teilgebiete anzugeben
    c. 2: Parameterwerte in einem zweidimensionalen Array, z. B. um die Blattflächenindices verschiedener Teilgebiete und Monate anzugeben
6. Datentyp, i. d. R. *float* oder *int*.

Die maximale Infiltrationsrate als skalarer Parameter ohne Standardwert wäre beispielsweise wie folgt definierbar (*np* ist eine gängige Abkürzung für das Paket NumPy):

```
>>> self.standardParameters = {'infilt': (np.nan, 0., np.inf,  1, 0, float)}
```

1.13.1.3 calcStaticSecPars & calcDynamicSecPars

Beide Methoden werden von *initParameters* aufgerufen. *calcStaticSecPars* wird nur einmalig aufgerufen und definiert feststehende Attribute, wie beispielsweise die relativen Flächenanteile von Teilgebieten, welche aus gegebenen Absolutflächen zu ermitteln sind. *calcDynamicSecPars* wird zusätzlich nach jeder Änderung der primären Parameter aufgerufen, um beispielsweise die Unit-Hydrograph-Ordinaten eines Einzellinearspeichers nach Aktualisierung der Speicherkonstante anzupassen. Die sekundären Parameter werden an *parameters.statSecPars* bzw. *parameters.dynSecPars* angehängt. Die Modell-Basisklasse implementiert keine sekundären Parameter, weshalb die entsprechenden Methoden bei Bedarf zu überlagern sind.

1.13.1.4 setContrPars & getContrPars

setContrPars ermöglicht das Setzen neuer Parameterwerte. *getContrPars* gibt geänderte Parameterwerte beispielsweise zum Schreiben neuer Modell-Initialisierungsdateien (Abschnitt 1.9.4) aus. Modifikationen der Methode der Basisklasse sind i. d. R. nicht notwendig.

1.13.1.5 getSolverPars

Aktuell ist lediglich der in (Abschnitt 1.11.1) beschriebene numerische Integrationsalgorithmus im Framework implementiert. Zu dessen Steuerung übergibt die Methode *getSolverPars* feste Parameterwerte an *parameters.solverPars* als Attribute. Durch Überlagerung können andere Parameterwerte gesetzt werden. Parallel zur Umsetzung weiterer Integrationsalgorithmen ist *getSolverPars* flexibler zu gestalten.

1.13.1.6 initArrays & initStateSpaceArrays

Der Konstruktor *__init__* ruft die Methode *initArrays* unter Weitergabe der modellspezifischen Informationen auf. *initArrays* initialisiert sämtliche Zeitreihen-Arrays, setzt deren Anfangszustände (*setInitVals*) und initialisiert ggf. zusätzliche Arrays für die Speicherung von Zwischenergebnissen bei der Anwendung numerischer Integrationsalgorithmen (*initStateSpaceArrays*).

initArrays wird grundsätzlich ergänzt. Vor dem Aufruf der gleichnamigen Methode der Basisklasse sind die Dictionaries *inputArrays*, *storageArrays* und *fluxArrays* zu definieren. Deren Schlüssel sind die Namen der Eingabe-, Speicher- und Flussgrößen als *string*. Die zugeordneten Objekte sind Tupel mit zwei bzw. drei Einträgen:

1. Spaltenanzahl: Falls ein Array nicht grundsätzlich auf eine bestimmte Spaltenanzahl beschränkt ist, sollte eine Variable übergeben werden. So kann z. B. für einen Gesamtgebietsausfluss die 1 festgesetzt, für eine räumlich verteilte Größe wie die Bodenfeuchte dagegen eine vom Benutzer vorgegebene Anzahl an HRUs angeben werden.
2. Dimensionalität: Mögliche Belegungen 1 und 2 für ein- und zweidimensionale Arrays. Es ist zu berücksichtigen, dass ein Array unabhängig von den jeweiligen Benutzervorgaben immer entweder ein- oder zweidimensional sein muss, damit der Zugriff auf darin hinterlegte Werte eindeutig codierbar ist. Auf eindimensionale Arrays wird grundsätzlich über einen Index, auf zweidimensionale Arrays über zwei Indices zurückgegriffen. Nimmt ein Benutzer beispielsweise die Möglichkeit zur Aufteilung eines Einzugsgebietes in HRUs nicht wahr, sind betroffene Größen wie die Bodenfeuchte folglich in zweidimensionalen Arrays mit nur einer Spalte hinterlegt.
3. Boolescher Wert der angibt, ob ein zusätzliches *stateSpaceArray* angelegt werden soll. Das ist für Eingabegrößen nie und für Speichergrößen immer erforderlich. Entsprechend ist dieser dritte Tupel-Eintrag den Flussgrößen vorbehalten, für die der Wert *True* nur dann zu setzen ist, wenn eine rechnerische Abhängigkeit von Speichergrößen besteht.

Im Folgenden ein Beispiel für eine Niederschlagsreihe *P* als gebietsweite Eingangsgröße, eine nach Wald, Acker und Siedlung differenzierte Interzeptionsspeicherung *IC* sowie die potenzielle (nicht von *IC* abhängige) Interzeptionsverdunstung *PEic* und die tatsächliche (von *IC* abhängige) Interzeptionsverdunstung *AEic*:

```
>>> self.inputArrays = {'P': (1, 1)}
>>> self.storageArrays = {'IC': (3, 2)}
>>> self.fluxArrays = {'PEic': (3, 2, False),
>>>                    'AEic': (3, 2,   True)}
```

initStateSpaceArrays führt die Initialisierung der Zustandsraum-Arrays durch. Wird der bislang implementierte numerische Integrationsalgorithmus verwandt (siehe Abschnitt 1.11.1), besteht keine Notwendigkeit zur Modifikation der Methode der Basisklasse. Bei anderen Algorithmen wären die Siebenen der beiden folgenden Quelltext-Zeilen durch die maximale Häufigkeit zu ersetzen, mit der jeder Wert einer Fluss- und Speichergröße in einem Rechenschritt (nicht zu verwechseln mit der Zeitschrittweite) aufgrund von Zwischenschritten und Tests ermittelt wird:

```
>>> s.arrays.flux.initStateSpaceArrays(7, s.fortran)
>>> s.arrays.storage.initStateSpaceArrays(7, s.fortran)
```

Parallel zur Umsetzung weiterer Integrationsalgorithmen ist *initStateSpaceArrays* flexibler zu gestalten. Um von den original Zeitreihen-Arrays unterscheiden zu können, ist den Variablennamen der Zustandsraum-Arrays ein Unterstrich vorangestellt.

1.13.1.7 setInitVals & getInitVals

setInitVals setzt benutzerdefinierte oder aus vorherigen Simulationsrechnungen stammende Anfangsbedingungen. *getInitVals* gibt berechnete Systemzustände für spätere Rechnungen in Form eines Dictionary aus. Insofern die Anfangsbedingungen ausschließlich in Speichergrößen hinterlegbar sind, kann die Model-Basisklasse unverändert beibehalten werden. Ergänzungen sind notwendig, falls die Berechnung einer Flussgröße von Werten der Speicher- oder Flussgrößen zu vorherigen Zeitschritten abhängt. Das ist z. B. bei der Faltungsoperation des Unit-Hydrograph der Fall. Bei Wellenablaufverfahren wie dem Muskingum-Verfahren, die auf finiten Differenzen basieren, empfiehlt es sich die entsprechende Raum-Zeit-Matrix der Knotenpunkte als *storageArray* anzulegen. Damit entfällt die Notwendigkeit *setInitVals* und *getInitVals* zu ergänzen.

1.13.1.8 setInput, getOutput & modifyInputArrays

getOutput wird von der übergeordneten Element-Instanz aufgerufen und gibt die berechnete(n) Ausgabegröße(n) des Modells zurück. Die Methode ist, da sie auf modellspezifische Arrays zurückgreift, grundsätzlich zu überschreiben.

setInput übernimmt die berechnete(n) Eingabegröße(n) der „Oberlieger"-Modelle. Die Übergabe erfolgt durch die übergeordnete Element-Instanz. Die Methode muss nur überschrieben werden, falls das Modell einen Zufluss vorsieht, was z. B. bei Verfahren zur Berechnung des Wellenablaufes oder der Hochwasserschutzwirkung von Rückhaltebecken der Fall ist.

Im Gegensatz zu *setInput* übernimmt *modifiyInputArrays* nicht die von „Oberlieger"-Modellen berechneten Ausgaben, sondern extern ermittelte, neue (i. d. R. meteorologische) Eingangsgrößen und überschreibt die alten. So werden z. B. Sensitivitätsanalysen vereinfacht. Modifikationen der Methode der Basisklasse sind i. d. R. nicht notwendig.

1.13.1.9 reset

Nach erfolgter Simulation setzt *reset* die Speicher- und Flussgrößen auf null bzw. die Anfangswerte zurück. Eine Ergänzung der Methode der Basisklasse kann erforderlich sein, wenn die in Abschnitt 1.13.1.7 erwähnte Notwendigkeit zur „zeitübergreifenden" Berechnung von Flussgrößen besteht.

1.13.1.10 writeArrays

Wie in den Abschnitten 1.10.2 und 1.12.4 beschrieben gibt *writeArrays* die berechneten Zeitreihen der Fluss- und Speichergrößen aus. Modifikationen der Methode der Basisklasse sind i. d. R. nicht notwendig.

1.13.1.11 python2fortran & fortran2python

Beide Methoden sind nur für das Einbinden von mit Fortran programmierten Modellen relevant. Durch Aufruf von *python2fortran* übergibt die Python-Modellinstanz sämtliche Arrays und Parameter an die Instanz des entsprechenden Fortran-Moduls. Durch Aufruf von *fortran2python* werden die von der Instanz des Fortran-Moduls berechneten Fluss- und Speichergrößen der Python-Modellinstanz verfügbar gemacht. Eine Ergänzung der Methode der Basisklasse kann erforderlich sein, wenn teilweise über den angegebenen Simulationszeitraum hinaus gerechnet wird, wie ggf. bei Unit-Hydrograph-Faltungen der Fall.

1.13.1.12 run

Die Methode *run* startet die eigentliche Modellrechnung. Bei Nutzung eines in Fortran programmierten Modells, werden *python2fortran* vor sowie *fortran2python* nach der Simulation so aufgerufen, dass nur der relevante Ausschnitt der Reihen übergeben wird. *run* muss nicht verändert oder ergänzt werden. Allerdings sind von den darin aufgerufenen Methoden *runFortranAdhoc*, *runFortranStateSpace*, *runPythonAdhoc*, *runPythonStateSpace*, *runCythonAdhoc*, *runCythonStateSpace* grundsätzlich diejenigen zu überschreiben, für die eine entsprechende programmtechnische Umsetzung vorliegt. Beispielsweise ist die klassische Simulation entsprechend HBV_{96} unterteilt in die Manipulation der Eingangsdaten, die ad-hoc-Lösung der Differenzialgleichungen und die Faltung des Dreieck-Unit-Hydrographs:

```
>>> def runPythonClassic(self, idxDateStart, idxDateEnd):
>>>     self.correctInput(idxDateStart, idxDateEnd)
>>>     self.adhoc(idxDateStart, idxDateEnd)
>>>     self.UHconvolution(idxDateStart, idxDateEnd)
```

1.13.1.13 getShortcuts

getShortcuts übergibt eine Befehlszeile, deren Ausführung zur Erhöhung des Programmierkomforts standardisierte Abkürzungen auf die verschiedenen Array- und Parameter-Handler setzt. Zur Ausführung ist dem Funktionsaufruf der Befehl *exec* voranzustellen:

```
>>> exec self.getShortcuts()
```

Folgende Abkürzungen werden gesetzt:

ph	=	self.parameters
phSt	=	self.parameters.statSecPars
phDy	=	self.parameters.dynSecPars
phSo	=	self.parameters.solverPars
ih	=	self.arrays.input
fh	=	self.arrays.flux
sh	=	self.arrays.storage

Werden andere Abkürzungen gewünscht, ist die Methode der Basisklasse zu überschreiben oder eine alternative Methode zu definieren.

1.13.2 Numerische Integration

Die Model-Basisklasse bietet zahlreiche Funktionalitäten, um Dopplungen bei der Implementierung eines Modellkonzeptes sowohl in einer ad-hoc- als auch einer Zustandsraum-Variante zu vermeiden. Falls erforderlich, wird die Initialisierung der zusätzlichen Zustandsraum-Arrays, wie in Abschnitt 1.13.1.6 beschrieben, automatisch durchgeführt. Der Datenaustausch zwischen diesen und den Zeitreihen-Arrays wird durch die Methoden *getStorages* und *setStorages* der Basisklasse durchgeführt. Zu überschreiben sind lediglich *calcFluxes* und *calcStorages* sowie – falls hinsichtlich der Zustände der Speichergrößen Restriktionen bestehen – *corrStorages*.

1.13.2.1 calcFluxes

Die Methode *calcFluxes* berechnet Flussgrößen, wie z. B. den Ausfluss aus einem Einzellinearspeicher in Abhängigkeit dessen Füllstandes. Als Funktionsparameter erhält *calcFluxes* drei Indices:

1. *i*: gibt den aktuellen Zeitschritt der Zeitreihen-Arrays an
2. *iS:* gibt an, wo die relevanten Werte der Speichergrößen in den Zustandsraum-Arrays hinterlegt sind
3. *jF*: gibt an, wo die berechneten Werte der Flussgrößen in den Zustandsraum-Arrays zu hinterlegen sind

calcFluxes ist grundsätzlich zu überschreiben. Mit den in Abschnitt 1.13.1.13 aufgeführten Abkürzungen sowie der in Abschnitt 1.13.1.6 erläuterten Namenskonvention der Zustandsraum-Arrays ließe sich der gebietsweite Effektivniederschlag *Peff* wie folgt aus dem Gesamtniederschlag *Ptot* als Eingangsgröße sowie dem Verhältnis der Bodenfeuchte *SM* als Speicherzustand und der maximalen Bodenfeuchte *SMmax* als Parameter berechnen:

$$\text{>>> fh.\_Peff[jF] = ih.Ptot[i]} \cdot \frac{\text{sh.\_SM[iS]}}{\text{ph.SMmax}}$$

Es ist zu beachten, dass *calcFluxes* einzelne Flüsse und nicht die Nettoänderungen einzelner Speichergrößen bestimmt. Dies ist aus hydrologischer Sicht sinnvoll, da i. d. R. einzelne Flüsse und hierbei insbesondere der Ausfluss am Gebietsauslass interessieren. Es erfordert aber eine Implementierung numerischer Integrationsalgorithmen die programmtechnisch von der Lehrbuchform abweicht, welche zugeschnitten ist auf Probleme bei denen Zustandstrajektorien – beispielsweise Himmelskörperbahnen – die eigentliche Zielgröße sind.

1.13.2.2 averageFluxes & addFluxes

Numerische Integrationsalgorithmen wie das Runge-Kutta-Verfahren berechnen ausgehend von verschiedenen Stützstellen mehrere Werte für Flussgrößen. Diese werden anschließend gewichtet gemittelt, wofür die Methode *averageFluxes* zur Verfügung steht.

addFluxes dient der Übergabe von in Zustandsraum-Arrays hinterlegten Werten der Flussgrößen an die Zeitreihen-Arrays. Mehrmaliger Verweis auf denselben Zeitindex der Zeitreihen-Arrays führt nicht zum Überschreiben sondern zum Kumulieren entsprechend der Rechenschrittweite. So wird berücksichtigt, dass sich die Gesamtsumme eines Flusses über einen Zeitschritt aus den Teilergebnissen einzelner Rechenschritte zusammensetzen kann.

Diese Methoden der Basisklasse müssen nicht überschrieben werden und spielen für die Modellentwicklung keine direkte Rolle

1.13.2.3 calcStorages

Die Methode *calcStorages* berechnet Speichergrößen, wie z. B. den Füllstand eines Einzellinearspeichers nach Abzug des Ausflusses. *calcStorages* benötigt fünf Funktionsargumente:

1. *i*: Index der relevanten Eingangs- und Flussgrößen in den Zeitreihen-Arrays
2. *iS*: Index der Anfangswerte der Speichergrößen in den Zustandsraum-Arrays
3. *jS*: Index der zu berechnenden Endwerte der Speichergrößen in den Zustandsraum-Arrays
4. *iF*: Index der relevanten Flussgrößen in den Zustandsraum-Arrays
5. *step*: Rechenschrittweite

calcStorages ist grundsätzlich zu überschreiben. Mit den in Abschnitt 1.13.1.13 aufgeführten

Abkürzungen sowie der in Abschnitt 1.13.1.6 erläuterten Namenskonvention der Zustandsraum-Arrays ließe sich die Zunahme der Bodenfeuchte *SM* wie folgt aus dem eingelesenen Gesamtniederschlag *Ptot* und dem zuvor berechneten Effektivniederschlag *Peff* berechnen:

```
>>> sh._SM[jS] = sh._SM[iS] + step · (ih.Ptot[i] – fh._Peff[iF])
```

step ist hierbei als auf die Zeitschrittweite bezogene Relativgröße zu sehen. *Ptot* und *Peff* sollten in der Einheit mm pro Zeitschrittweite vorliegen. *step* nimmt somit sinnvollerweise nur Werten größer null bis maximal eins an.

1.13.2.4 getStorages, setStorages & moveStorages

Die Methoden *getStorages* und *setStorages* regeln Wertübergaben zwischen Zeitreihen- und Zustandsraum-Arrays der Speichergrößen. *moveStorages* verschiebt Werte der Speichergrößen von einer Position der Zustandsraum-Arrays zu einer anderen. Diese Methoden der Basisklasse müssen nicht überschrieben werden und spielen für die Modellentwicklung keine direkte Rolle.

1.13.2.5 calcError1

Die Methode *calcError1* berechnet ein Maß für die Bewertung der Genauigkeit der numerischen Integration und wird zur dynamischen Festlegung der Rechenschrittweite oder des Integrationsalgorithmus genutzt (siehe Abschnitt 1.11.1). Die als Funktionsparameter übergebenen Zustandsraum-Indices *iS* und *jS* verweisen auf zwei in unterschiedlicher Weise erzielte Integrationsergebnisse. Für sämtliche Speichergrößen werden deren Betragsdifferenzen ermittelt und das Maximum an die Variable *error1* übergeben. Das Überschreiben der Methode ist nur erforderlich, wenn eine alternative Fehlerberechnung gewünscht ist (z. B. eine Gewichtung der Differenzbeträge entsprechend der Flächengröße einzelner Teilgebiete vor Bestimmung des Maximums).

1.13.2.6 corrStorages, corrMin, corrMax, corrMinMax

Die Methode *calcStorages* wird für Modelle benötigt, deren Speichergrößen nur einen bestimmten Wertebereich annehmen dürfen (siehe Abschnitt 1.11.1). Gängigstes Beispiel ist die Nichtnegativitäts-Bedingung. Enthält eine Modellkonzeption derartige Restriktionen, ist die Methode der Basisklasse zu überschreiben. Zum einen setzt *corrStorages* Werte von Speichergrößen der Zustandsraum-Arrays, die beim Index *iS* hinterlegt sind und außerhalb des zulässigen Wertebereiches liegen, auf den über- oder unterschrittenen zulässigen Grenzwert zurück. Zum anderen wird, bei Belegung des zweiten Funktionsparameters *updateError2* mit dem Wert eins, der größte Betrag aller Restriktionsverletzungen an die Variable *error2* übergeben. *error2* ist somit wie die in Abschnitt 1.13.2.5 beschriebene Größe *error1* für die Bewertung der Güte der numerischen Integration heranzuziehen.

Zur übersichtlichen Programmierung von *corrStorages* stehen die Hilfsfunktionen *corrMin*,

corrMax und *corrMinMax* zur Verfügung. Die Korrektur der Bodenfeuchte *SM* mit dem Wert Null als unterer und der maximalen Bodenfeuchte *SMmax* als oberer Grenze ließe sich wie folgt notieren:

```
>>> sh._SM[iS] = self.corrMinMax(sh._SM[iS], 0., ph.SMmax, updateError2)
```

Es ist zu beachten, dass *error2* vor allen Aufrufen von *corrMin*, *corrMax* und *corrMinMax* auf null zurückzusetzen ist, falls die Aktualisierung von *error2* gefordert wird:

```
>>> if updateError2 == 1:
>>>     phSo.error2 = 0.
```

1.13.2.1 adaptiveEulerRungeKutta, euler & rungeKutta

Die numerische Integration der Differenzialgleichungen eines Modells kann bislang nur mit der Methode *adaptiveEulerRungeKutta* erfolgen. *adaptiveEulerRungeKutta* steuert die Werteübergaben zwischen Zeitreihen- und Zustandsraum-Arrays und führt die Integration für alle Zeitschritte durch. Hierbei wird jeweils zwischen dem expliziten Euler- und dem expliziten Runge-Kutta-Verfahren gewählt, d. h. entweder die Hilfsfunktion *euler* oder die Hilfsfunktion *rungeKutta* aufgerufen sowie die geeignete Rechenschrittweite festgelegt. Diese Methoden der Basisklassen müssen nicht überschrieben werden und spielen für die Modellentwicklung keine direkte Rolle. Inhaltliche Aspekte der Methoden sind im Abschnitt 1.11 beschrieben.

1.13.3 Eliminierung unstetiger Differenzialgleichungen

Die Verwendung von Grenzwerten in den Differenzialgleichungssystemen hydrologischer Modelle erschwert die Anwendung numerischer Integrationsalgorithmen (siehe Abschnitt 1.11.1). Darüber hinaus sind nach Kavetski & Kuczera (2007) sämtliche Formen von Unstetigkeit in Modellgleichungen potenzielle Quellen für Unregelmäßigkeiten und lokale Optima in Fehleroberflächen, die ihrerseits die Parameterkalibrierungen erschweren. Die Model-Basisklasse enthält daher mehrere Funktionen zur „Glättung“ dieser Unstetigkeiten, deren Glättungsgrad über Parameter einstellbar ist (Abschnitt 1.13.3.1). Zudem stehen Methoden zur Verfügung, welche die Einheitlichkeit und Anschaulichkeit der in einem Modell eingesetzten Glättungsparameter sicherstellen sollen (Abschnitt 1.13.3.2).

1.13.3.1 Glättungsfunktionen

Die drei bislang im Framework implementierten Glättungsfunktionen sind in Abbildung 1.8 veranschaulicht. Die unstetige Originalfunktion ist jeweils in schwarz dargestellt. Die Linien in rot, grün und blau ergeben sich durch unterschiedliche Belegung des jeweiligen Glättungsparameters. Wie die roten Linien andeuten, lassen sich die Originalfunktionen durch geringe Glättungen beliebig genau approximieren. Die blauen Linien zeigen die Ergebnisse starker Glättungen, die bis hin zur annähernden Linearisierung des Zusammenhangs im relevanten Wertebereich möglich sind. Nicht nur die Glättungsfunktionen selbst, sondern auch deren (höhere) Ableitungen sind stetig. Die drei implementierten Funktionen sollten ausreichen, um einen Großteil der in hydrologischen Modellgleichungen auftretenden Unstetigkeiten eliminieren zu können.

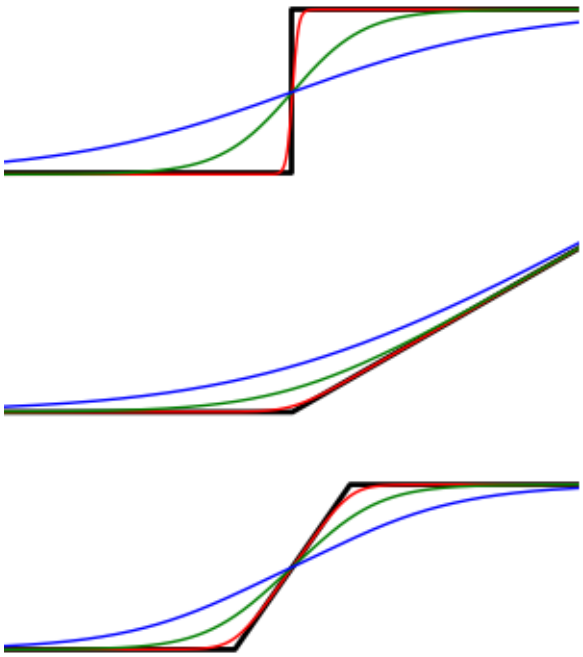


Abbildung 1.8: Glättungsfunktionen

Durch Belegung der Glättungsparameter mit kleinen Werten ist eine beliebige Annäherung an den Originalzusammenhang möglich. Mit Zunahme der Krümmungen im Bereich der vorherigen Unstetigkeiten wird allerdings die numerische Integration erschwert. Dies kann zu einer erhöhten Rechendauer bzw. zum Verfehlen der gewünschten Genauigkeit führen. Zudem sind bei expliziten Algorithmen Instabilitäten nicht auszuschließen. Entsprechend sollten seitens des Modellentwicklers Minimalwerte der Glättungsparameter empfohlen oder besser wie in Abschnitt 1.13.1.2 beschrieben direkt in die jeweilige Modell-Klasse implementiert werden.

Die erste Glättungsfunktion *smooth1* (Abbildung 1.8, oben) ersetzt „Schalterfunktionen“ entsprechend Gleichung 1.1. Bedingungen dieser Art legen beispielsweise fest, ob Niederschlag aufgrund der Unter- oder Überschreitung einer Grenztemperatur in fester oder flüssiger Form fällt.

$$f_{orig1}(x) = \begin{cases} 0 & wenn \quad x \le 0 \\ 1 & wenn \quad x > 0 \end{cases} \qquad 1.1$$

smooth1 basiert auf der logistischen Funktion (Gleichung 1.2). Durch die darin enthaltene Exponentialfunktion besteht die Gefahr des Variablenüberlaufs (der Python Datentyp *float* für Gleitkommazahlen kann maximal den Wert 1,7976931348623157e+308 annehmen und entspricht damit beispielsweise in C dem Datentyp *double*). Das macht die in Gleichung 1.2 enthaltene Abfrage erforderlich. Im Rahmen der Maschinengenauigkeit führt die Abfrage aber zu keiner Unstetigkeit.

$$f_{smooth1}(x, s_1) = \begin{cases} 1 - \dfrac{1}{1 + \exp(x/s_1)} & wenn \quad x < \ln(float_{\max}) \cdot s_1 \\ 1 & wenn \quad x \geq \ln(float_{\max}) \cdot s_1 \end{cases} \tag{1.2}$$

Durch Integration der Gleichung 1.1 erhält man Gleichung 1.3 (Abbildung 1.8, mittig). Dieser Typ von Gleichung liegt beispielsweise dem Grad-Tag-Verfahren zugrunde, bei dem ab der Überschreitung der Grenztemperatur eine zur Differenz aus Temperatur und Grenztemperatur proportionale Schneeschmelze erfolgt, falls ausreichend Schnee liegt.

$$f_{orig2}(x) = \begin{cases} 0 & wenn \quad x \leq 0 \\ x & wenn \quad x > 0 \end{cases} \tag{1.3}$$

Analog zum unstetigen Fall lässt sich die in Gleichung 1.4 definierte Funktion *smooth2* durch Integration von Gleichung 1.2 ermitteln.

$$f_{smooth2}(x, s_1) = \begin{cases} s_2 \cdot \ln(1 + \exp(x/s_2)) & wenn \quad x < \ln(float_{\max}) \cdot s_1 \\ x & wenn \quad x \geq \ln(float_{\max}) \cdot s_1 \end{cases} \tag{1.4}$$

Die beiden ersten Glättungsfunktionen werden in vielen Bereichen standardmäßig angewandt (die logistische Funktion z. B. in Künstlichen Neuronalen Netzen) und werden von Kavetski & Kuczera (2007) für die hydrologische Modellierung empfohlen. Problematischer ist der in Gleichung 1.5 angegebene Fall (Abbildung 1.8, unten). Derartige Zusammenhänge treten beispielsweise auf, wenn die Aufteilung von Niederschlag in Schnee und Regen anhand zweier Grenzwerte erfolgt: Schnee unterhalb des unteren Grenzwertes, Regen oberhalb des oberen Grenzwertes und dazwischen eine linear interpolierte Mischung beider Niederschlagsformen.

$$f_{orig3}(x) = \begin{cases} 0 & wenn \quad x \leq -1/2 \\ x - 1/2 & wenn \quad -1/2 < x < 1/2 \\ 1 & wenn \quad x \geq 1/2 \end{cases} \tag{1.5}$$

Kavetski & Kuczera (2007) empfehlen zur Lösung solcher und komplexerer Probleme eine Zerlegung in Teilprobleme. So wäre Gleichung 1.5 unterhalb und oberhalb vom x-Wert Null jeweils durch Varianten von Gleichung 1.4 zu ersetzen. Diese beliebig erweiterbare Vorgehensweise hat den Nachteil, dass die (maschinengenaue) Stetigkeit der zusammengesetzten Funktion nur bei geringen Glättungen gegeben ist. Das schränkt den Spielraum des Modellanwenders ein und schließt die in Abschnitt 1.13.3.2 beschriebene Strategie zur einheitlichen Parametrisierung sämtlicher Glättungsfunktionen aus. Im Framework wurde daher stattdessen der in Abbildung 1.9 veranschaulichte und in Gleichung 1.6 notierte Ansatz gewählt. *smooth3* setzt sich, dem Vorschlag von Kavetski & Kuczera (2007) entsprechend, aus zwei Glättungsfunktionen entsprechend *smooth2* (Gleichung 1.4) zusammen (blaue Linien). Diese werden aber entsprechend *smooth1* (Gleichung 1.2) gewichtet (grüne Linien, Wichtungsfaktor *w* in den Gleichungen 1.6 und 1.7). Die Endergebnisse von *smooth3* (rote Linien) zeigen sowohl eine starke Annäherung an den Originalzusammenhang bei geringen Glättungsgraden (oberes Bild) als auch saubere Übergänge ohne Unstetigkeiten bei mittleren und großen Glättungsgraden (mittleres und unteres Bild).

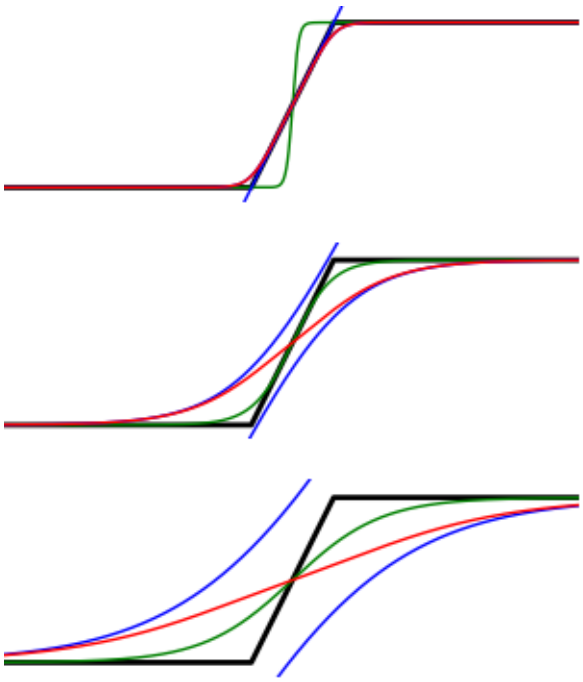


Abbildung 1.9: Zerlegung von *smooth3*

$$f_{smooth3}\left(x,\ s_3\right) = \left(1-w\right)\cdot f_{smooth2}\left(x,\ s_2\right) + w\cdot\left(1-f_{smooth2}\left(1-x,\ s_2\right)\right) \qquad 1.6$$

$$w = f_{smooth1}\left(x-1/2,\ s_1\right) \qquad 1.7$$

Der Glättungsparameter s_2 in Gleichung 1.6 entspricht s_3 (Gleichung 1.8). Folglich verhält sich *smooth3* in den Randbereichen wie *smooth2*.

$$s_2 = s_3 \qquad 1.8$$

Komplizierter ist die Wertbelegung von s_1, wofür der Zusammenhang in Gleichung 1.9 ermittelt wurde. Die Berechnung von s_1 mit der Potenzgleichung bewirkt, dass die dritte Ableitung von Gleichung 1.6 beim x-Wert 0,5 annährend null ist (keine Krümmung der Steigung im Zentrum). So werden drei Dinge sichergestellt: (1) im zentralen Bereich treten keine Verflachungen auf, (2) in den Randbereichen erfolgt eine rasche Annährung an den Funktionsverlauf von *smooth2* und (3) die Funktion ist (im Bereich der *smooth2* betreffenden Maschinengenauigkeit) streng monoton steigend. Die zusätzliche Restriktion in Gleichung 1.9 greift im Bereich extrem geringer Glättungen, wo die Annäherung der dritten Ableitung an Null problematisch ist. Da hierbei aber die beiden *smooth2*-Verläufe auch ohne Gewichtung beinahe fließend ineinander übergehen, macht sich die Verwendung dieser Sicherheitsschranke nicht nachteilig bemerkbar.

$$s_1 = \begin{cases} 0{,}54 \cdot s_2^{1{,}17} & wenn \quad s_2 > 0{,}0724 \\ 0{,}025 & wenn \quad s_2 \le 0{,}0724 \end{cases} \qquad 1.9$$

In jeder Glättungsfunktion ist null als x-Wert der ursprünglichen Unstetigkeit angesetzt, in *smooth3* zudem der x-Wert eins für die zweite Unstetigkeit. Weichen die unstetigen Originalfunktionen hiervon ab, sind entsprechende Normierungen im Funktionsaufruf durchzuführen. Die Steigungen bzw. Spannweiten der Glättungsfunktionen betragen eins, entsprechend ist das Funktionsergebnis mit dem gewünschten Wert der Steigung oder Spannweite zu multiplizieren. Der Wert null wird stets bei kleinen x-Werten angenähert, durch Subtraktion des Funktionsergebnisses von eins wird dieses Verhalten umgekehrt.

1.13.3.2 Glättungsparameter

Im Prinzip ließe sich jede der in Abschnitt 1.13.3.1 eingeführten Glättungsfunktionen bei jeder Anwendung auf die i. d. R. zahlreichen Unstetigkeiten in den Gleichungen hydrologischer Modelle getrennt parametrisieren. Das scheint nur dann sinnvoll, wenn die Wertebelegung der Glättungsparameter inhaltlich begründet a-priori erfolgen kann, beispielsweise wenn ein Zusammenhang zur räumlichen Variabilität einer Gebietseigenschaft besteht. Ansonsten empfiehlt sich eine Rückführung der einzelnen Glättungsparameter auf wenige intuitiv verständliche Metaparameter. Über diese kann der Anwender den „globalen“ Glättungsgrad des Differenzialgleichung-Systems festlegen.

Die Model-Basisklasse stellt hierfür die Methoden *calcSmoothPar1* und *calcSmoothPar2* bereit. Beide erwarten einen Metaparameter s_m. Dieser entspricht dem horizontalen Abstand zur jeweiligen Unstetigkeit, bei dem die Differenz zwischen Original- und Glättungsfunktion ein Prozent beträgt. Ein Beispiel: Im unstetigen Fall entsteht Stammabfluss nur, wenn die maximale Interzeptionsspeicherung von 5 mm überschritten ist. Unter Rückgriff auf *smooth1* ist die obere Grenze der Interzeptionsspeicherung von 5 mm dagegen derjenige Speicherinhalt, bei dem 50 % des Niederschlages zurückgehalten werden. Bei einem s_m von 1 mm beträgt der Rückhalt bei 4 mm Speicherinhalt 99 % und bei 6 mm 1 %. Umgekehrt beträgt der Stammabfluss bei 4 mm Speicherinhalt 1 % und bei 6 mm 99 %. Bei einem s_m von 0,1 mm nimmt die Interzeptionsspeicherung nur selten Werte geringfügig größer als 5,1 mm an und beide Ansätze liefern sehr ähnliche Systemreaktionen.

Für *smooth1* ist die Beziehung zwischen dem Metaparameter s_m und dem Effektivparameter s_1 in Gleichung 1.10 definiert. Für positive Werte lässt diese sich nach Gleichung 1.11 umstellen, welche in der Methode *calcSmoothPar1* implementiert ist.

$$0{,}01 = 1 - \frac{1}{1 + \exp\left(s_m / s_1\right)} \qquad 1.10$$

$$s_1 = \ln\left(100 - 1\right)^{-1} \cdot s_m \qquad 1.11$$

Analog wäre für *smooth2* Gleichung 1.12 nach s_2 umzustellen, wofür kein allgemeiner Ausdruck für den relevanten Wertebereich existiert. Stattdessen wird in der Methode *calcSmoothPar2* fallbezogen ein Näherungswert nach Gleichung 1.13 berechnet, der dem iterativen

Newton-Raphson-Verfahren als Startwert dient. Der hierfür benötigte Gradient ist durch die partielle Ableitung von Gleichung 1.10 nach s_2 gegeben (Gleichung 1.14).

$$0{,}01 = s_2 \cdot \ln\left(1 + \exp\left(s_m / s_2\right)\right) \quad 1.12$$

$$s_2 = 0{,}3 \cdot s_m^{0{,}84} \quad 1.13$$

$$\frac{\partial}{\partial s_2}\left(s_2 \cdot \ln\left(1 + \exp\left(s_m / s_2\right)\right)\right) = \frac{s_m}{s_2 \cdot \exp\left(s_m / s_2\right) + s_2} \cdot \ln\left(\exp\left(-s_m / s_2\right) + 1\right) \quad 1.14$$

Da *smooth3* im Randbereich etwa denselben Verlauf wie die jeweils relevante *smooth2*-Variante zeigt, ist s_3 äquivalent zu s_2 und kann ebenfalls mit *calcSmoothPar2* berechnet werden. Bei Glättungsgraden die in Bezug auf die Spannweite der beiden ursprünglichen Unstetigkeiten relativ groß sind, können geringfügige Abweichungen von der 1 %-Vorgabe auftreten.

1.13.4 Fortran

In der Regel sollte jede Modellentwicklung zunächst in Form von reinem Python-Quelltext erfolgen. Vorteile von Python wie dessen Interaktivität und insbesondere dessen umfangreiches automatisches Exception-Handling erleichtern das Programmieren und Testen. Nach Abschluss der Python-Umsetzung des Modells wird dessen Fortran-Äquivalent erstellt. Dieses umfasst nur die laufzeitkritischen Funktionen, welche unmittelbar mit Modellrechnungen in Verbindung stehen. Zum Beispiel wird das Einlesen und Ausschreiben von Zeitreihen auch bei Verwendung eines Fortran-Modells von in Python implementierten Methoden durchgeführt. Entsprechend ergeben sich für den Nutzer kaum Unterschiede in der Handhabung von Python- und Fortran-Modellen. Die vielleicht einzige Ausnahme hiervon ist, dass mit Verwendung einer Fortran-Implementierung das automatische Exception-Handling eingeschränkt wird. Durch fehlerhafte Manipulation einer Model-Instanz (Veränderungen der Dimensionalität von Arrays u. Ä.) vor Beginn der Simulationsrechnung können somit fehlerhafte Speicherzugriffe und Programmabstürze erfolgen, was lediglich denjenigen (erfahrenen) Anwender betreffen sollte, der den Funktionsumfang des Frameworks erweitert.

Zur Anfertigung eines Fortran-Modells ist zunächst ein (beliebiges) Objekt des neuen Modells im *stateSpace*-Modus zu instanziieren (bzw. ist die Instanziierung vom Framework vornehmen zu lassen). Anschließend werden die von der Model-Klasse vererbten Methoden *printFortranSourceCode* und *compileFortranSourceCode* aufgerufen.

1.13.4.1 Erstellen von Quelltext

Die Methode *printFortranSourceCode* der Model-Basisklasse schreibt den relevanten Python-Quelltext des jeweils umgesetzten Modells im Fortran 90-Standard aus. Die Textdatei wird im Ordner *temp* (siehe Abschnitt 1.8) unter dem Namen der Modellklasse gefolgt von der Endung *.f90* gespeichert. Alternativ kann eine andere Pfadbezeichnung als String über den Funktionsparameter *filename* übergeben werden. Als Ersatz für die Objektstruktur des Python-Modells werden die für die Simulationsrechnung benötigten Daten in einem Modul gekapselt. Die später in

Python eingebundene Instanz eines solchen Moduls ermöglicht so die Entkopplung von Datenübergaben und Funktionsaufrufen. Für den Anwender ergibt sich hieraus der Vorteil, dass mit einem solchen Fortran-Objekt wie mit einem Python-Objekt interaktiv gearbeitet werden kann, was bei der Analyse eventueller Ergebnisdifferenzen beider Implementierungen hilfreich ist.

Standardmäßig werden diejenigen notwendigen Methoden als Fortran-Quelltext ausgeschrieben, die von der Model-Basisklasse unverändert übernommen werden können, wie *addFluxes* oder *getStorages*. Zudem wird vom Benutzer für den Funktionsparameter *functions* eine Liste mit Referenzen auf alle modellspezifischen Methoden erwartet. Vorausgesetzt, die erste Element-Instanz enthält eine Python-Implementierung von HBV_{96}, wird der entsprechende Fortran-Quelltext wie folgt erstellt:

```
>>> m = HWU('rhein').elements[0].model
>>> m.printFortranSourceCode([m.correctInput, m.adhoc, m.UHconvolution])
```

Die erstgenannten, vererbten Methoden sollten stets als unmittelbar kompilierbarer Fortran-Quelltext ausgegeben werden. Das ist bei benutzerdefinierten Methoden nicht grundsätzlich der Fall, z. B. bei Verwendung von Python-Befehlen, für die kein Fortran-Äquivalent existiert oder die in den Konversationsroutinen bislang nicht berücksichtigt wurden. In solchen Situationen kann der Nutzer die ausgeschriebene Fortran-Datei entweder manuell verändern oder Übersetzungsbefehle in den Python-Quelltext aufnehmen. Letzteres ist empfehlenswerter, da es Dopplungsaufwand beim mehrmaligen Ausschreiben von Fortran-Dateien umgeht. Übersetzungsbefehle werden anhand der Zeichenkombination *#@* kenntlich gemacht. Folgende Möglichkeiten bestehen:

1. *#@skipLine*: Der Befehl steht i. d. R. am Ende eine Quelltext-Zeile und führt dazu, dass diese nicht übersetzt wird.
2. *#@skipLines*: Es sind jeweils zwei Befehle zu setzen, die i. d. R. ganze Quelltext-Zeilen einnehmen. Alle Zeichen dazwischen werden nicht übersetzt.
3. *#@localReplace*: Steht am Ende einer Quelltext-Zeile und wird gefolgt von Paaren „alten“ und „neuen“ Quelltextes, deren Start und Ende ebenfalls über die *#@*-Zeichenkombination kenntlich zu machen ist.
4. *#@globalReplace*: Es sind jeweils zwei Befehle zu setzen, die i. d. R. ganze Quelltext-Zeilen einnehmen. Die Ersetzungspaare sind dazwischen wie für *#@localReplace* beschrieben zu definieren. Die Ersetzung findet im gesamten Quelltext der Methode statt.
5. *#@int*: Alle dem Befehl folgenden, mit Komma getrennten Variablennamen werden als Ganzzahl deklariert (Python: *int*, Fortran: *integer*).
6. *#@float*: Alle dem Befehl folgenden, mit Komma getrennten Variablennamen werden als Gleitkommazahl deklariert (Python: *float*, Fortran: *real(8)*).

Der Befehl *#@int* ist selten notwendig da *printFortranSourceCode* eine automatische Typerkennung von Konstanten und Variablen enthält. Liefert keiner der Algorithmen zur Typerkennung ein Ergebnis, wird der deutlich häufigere Gleitkommazahl-Typ angenommen. Fehldekla-

rationen können insbesondere dann auftreten, wenn eine Variable nicht direkt in der übersetzten Methode verwendet, sondern lediglich an eine Submethode weitergereicht wird. Der Fehler wird vom Fortran-Compiler festgestellt und ausgegeben, woraufhin der korrekte Datentyp durch den Befehl *#@int* erzwungen werden kann. Da die automatische Typerkennung im Zweifelsfall Gleitkommazahlen favorisiert, ist das Setzen von *#@float* i. d. R. nicht notwendig. Es kann aber zur Strukturierung des Quelltextes oder zur Umgehung von (bislang nicht aufgetretenen) Ganzzahl-Fehldeklarationen dienen.

Das folgende Beispiel veranschaulicht die Befehle zur Auslassung und Ersetzung. Im gesamten Quelltext wird *python* durch *fortran* sowie *2.7* durch *90* ersetzt, unabhängig von den jeweils benachbarten Zeichen. Zeile 5 sowie die Zeilen 8 und 9 werden ignoriert. In Zeile 12 wird der Index *0* durch *1* ersetzt.

```
(1)  >>> #@globalReplace
(2)  >>> #@python#@ -> #@fortran#@
(3)  >>> #@2.7#@ -> #@90#@
(4)  >>> #@globalReplace
(5)  >>> a = 1   #@skipLine
(6)  >>> language = 'python'
(7)  >>> #@skipLines
(8)  >>> b = 2
(9)  >>> c = 3
(10) >>> #@skipLines
(11) >>> version = '2.7'
(12) >>> firstValue = allValues[0]    #@localReplace #@0#@ -> #@1#@
```

Aufgrund sämtlicher Übersetzungsbefehle ist der obige Python-Quelltext gleichbedeutend mit dem unten stehenden. Besonders beachtet werden sollte Zeile 12 (oben) bzw. 3 (unten). In Python ist der kleinste Index-Wert null, in Fortran dagegen eins. Die Konversionsroutinen nehmen hierauf bei der Wertebelegung von Laufvariablen in *range*- bzw. *do*-Schleifen Rücksicht. Bei der festen Vorgabe eines Index-Wertes muss aber (bislang) ein entsprechender *#@localReplace*-Befehl angefügt werden. Ein analoges Vorgehen kann z. B. auch bei der Verwendung von berechneten Indices nötig sein.

```
(1)  >>> language = 'fortran'
(2)  >>> version = '90'
(3)  >>> firstValue = allValues[1]
```

Neben den unterschiedlichen Start-Indices sequentieller Datentypen ist zu beachten, das Python zwischen Groß- und Kleinschreibung unterscheidet, Fortran aber nicht. Standardmäßig ist der Zugriff von Python auf die verschiedenen Variablen und Funktionen des späteren Fortran-Moduls nur über Kleinbuchstaben möglich. Hieraus können Namenskonflikte resultieren. Ein Beispiel ist der HBV_{96}-Wasserfluss *IN* (siehe Gleichung 2.19 ff.) Der Variablenname *IN* ist in Python zulässig. Dagegen ist *in* ein Python-Standardoperator, was den pythonseitigen Zugriff auf die Variable *in* eines Fortran-Moduls ausschließt. Für solche Fälle wird als Konvention vor-

geschlagen, an den ursprünglichen Variablennamen einen Unterstrich anzuhängen, also *IN_* bzw. *in_* zu schreiben.

Die Methode *printFortranSourceCode* und ihre Unterfunktionen sind bislang, zumindest bzgl. der modellspezifischen Methoden, eine Mischung aus gut strukturierter Programmierung und ad-hoc-Lösungsansatz für bisher aufgetretene Konversionsprobleme. Entsprechend sind fehlerhafte Konversionen bei neuartigen Problemen nicht auszuschließen. Konversionsfehler führen i. d. R. bereits beim anschließenden Kompilieren zu Fehlermeldungen und zum Abbruch, woraufhin dem Python-Quelltext entsprechende Übersetzungsbefehle hinzuzufügen oder Korrekturen am Fortran-Quelltext vorzunehmen sind. Dennoch empfiehlt es sich, am Ende jedes Entwicklungsprozesses Simulationsergebnisse der Fortran-Implementierung mit denen der Python-Implementierung zu vergleichen. Im fehlerfreien Fall treten nur marginale Unterschiede „im hinteren Nachkommastellenbereich“ aufgrund von Typkonversionen auf.

1.13.4.2 Kompilieren und Einbinden von Quelltext

Die Methode *compileFortranSourceCode* der Model-Basisklasse steuert den „Fortran to Python interface generator“ (F2PY), welcher Bestandteil von NumPy ist. F2PY modifiziert den zuvor erstellten Fortran-Quelltext, führt dessen Kompilierung durch, erstellt C-Quelltext zur Einbindung von Fortran über das Application Programmers Interface (Python/C API), kompiliert auch diesen und erstellt letztendlich ein „pyd“-Modul, das weitestgehend einer dll-Datei entspricht und über den Befehl *import* in Python geladen wird.

Der Methode *compileFortranSourceCode* können Dateinamen und Verzeichnispfade übergeben werden, falls von den Vorgaben abgewichen werden soll. Zudem können durch Übergabe von Strings an die Funktionsparameter *fortranCompiler* und *cCompiler* die zu verwendenden Fortran- und C-Compiler festlegt werden. In der bisherigen Modellentwicklung wurde auf Absoft ProFortran 13.0 und Microsoft Visual Studio 9.0 zurückgegriffen. Eine Liste der geeigneten Compiler findet sich in der paketinternen F2PY-Dokumentation. Das Vorhandensein aller relevanten Compiler- und Python-Systemvariablen ist sicherzustellen (z. B. liegt die Datei *f2py.py* im Python-Unterordner *Scripts*).

Der Python-Interpreter kann nur eine Instanz des von F2PY erstellten Moduls referenzieren, d. h. alle Instanzen derselben Model-Klasse teilen sich dieselbe Modul-Instanz. Entsprechend ruft die Model-Methode *run* (Abschnitt 1.13.1.12) vor jeder Simulationsrechnung *python2fortran* und nach jeder Simulationsrechnung *fortran2python* auf (Abschnitt 1.13.1.11), um die globalen Moduldaten auszutauschen. Hiermit ist eine gewisse Rechenzeit verbunden, die im Vergleich zur eigentlichen Simulationsdauer i. d. R. gering ausfällt. Sollte dies bei Spezialanwendungen – z. B. bei Zustandsnachführungen – anders sein, bietet sich der multiprocessing-Modus an (Abschnitt 1.7). Hierbei ist jede Model-Instanz in einem separaten Prozess mit jeweils eigener Instanz des Fortran-Moduls enthalten. Entsprechend kann im multiprocessing-Modus der Funktionsumfang von *python2fortran* sowie *fortran2python* modifiziert werden, um nicht grundsätzlich alle globalen Moduldaten auszutauschen, sondern nur die jeweils relevanten oder veränderten Daten.

2 Implementierte Modelle

Die Implementierung von HBV_{96} ist fertig gestellt, die von $LARSIM_{ME}$ geplant.

2.1 HBV96 (ad hoc-Strategie)

Die in diesem Abschnitt aufgeführten Prozessgleichungen sind im Modellframework HydPy umgesetzt und basieren weitgehend auf dem HBV_{96}-Handbuch (SMHI, 2011) und dem Abschlussbericht zur Verbesserung der HBV_{96}-Parametrisierung im Vorhersagesystem der BfG (SMHI, 2009). Da die vorhandenen Quellen nicht hinreichend detailliert sind, folgt die Modellumsetzung in einigen Aspekten der Interpretation des Autors. Betroffen hiervon ist insbesondere die Reihenfolge der einzelnen Rechenschritte. Von einer weitgehenden Übereinstimmung mit der originalen HBV_{96}-Implementierung im Integrated Hydrological Modelling System (IHMS) ist aufgrund der guten Übereinstimmung interner Modellzustände und der Modellausgabe auszugehen (siehe Abschnitt 2.1.8). Bei Widersprüchen zwischen den Simulationsergebnissen der IHMS-Implementierung und den genannten Quellen hat die Nähe zu den IHMS-Ergebnissen Priorität. Daraus ergibt sich beispielsweise die von SMHI (2009) abweichende Gleichung 2.36 für die schnelle Abflusskomponente. Modelloptionen sind i. d. R. nicht berücksichtigt, falls in aktuellen und absehbaren HBV-Umsetzungen der BfG kein Rückgriff auf diese stattfindet bzw. geplant ist. Dies betrifft beispielsweise die Zonenunterteilung zur Berücksichtigung ungleichmäßiger Schneeverteilungen (siehe Abschnitt 2.1.3). Vereinzelt wurden von SMHI (2011) abweichende Variablenbezeichnungen gewählt. Um beispielsweise eine Verwechslung der maximalen Perkolation *perc* mit der tatsächlichen Perkolation *PERC* zu verhindern wird erstere *PERCMAX* genannt.

Nach der HBV_{96}-Konzeption ist ein Flussgebiet in mehrere, nur durch Flussläufe verbundene Teilgebiete unterteilt. Um unterschiedliche Charakteristika eines Teilgebietes abzubilden, lassen sich diese weiter in Zonen unterteilen. Vier Zonentypen sind der Gebietsfläche zugeordnet (Abbildung 2.1). Die Typen „Field“ und „Forest“ sind hinsichtlich ihrer Prozessgleichungen identisch. Abweichendes Systemverhalten lässt sich durch unterschiedliche Belegungen bestimmter Parameterwerte wie der maximalen Interzeptionsspeicherung erzielen. In Zonen vom Typ „Glacier“ werden Interzeptions- und Verdunstungsprozesse vernachlässigt und das Schneemodul der „Field“- und „Forest“-Zonen um die Gletscherschmelze erweitert. „Internal Lake“-Zonen sind als oberflächliche Fortsetzung des tiefen Grundwassers der übrigen Zonen zu verstehen. Jeder der genannten Zonentypen kann mehrfach in einem Teilgebiet auftreten, allerdings nur mit denselben Parameterwerten. Unterschiede ergeben sich dagegen in den meteorologischen Randbedingungen aufgrund der verschiedenen Höhenniveaus einzelner Zonen und der Höhenkorrektur der Eingabereihen. Der fünfte Zonentyp „Outlet Lake“ stellt einen See bzw. Speicher am Teilgebietsauslass dar und ist in der HydPy-Implementierung bislang nicht umgesetzt. Die Wellenablaufberechnung ist an das Muskingum-Verfahren angelehnt (McCarthy, 1939).

Die in Abbildung 2.1 dargestellten Gleichungen zeigen die Umsetzung des variable source area-Ansatzes, dessen Einführung die größte Änderung des HBV_{96}-Modells nach dessen Erstveröffentlichung darstellt (Lindström et al., 1997). Die Anbindung der schnellen Abflusskomponente an den mittleren Bodenfeuchtegehalt, aber auch der kapillare Wiederaufstieg in die verschiedenen Zonen aus dem gemeinsamen oberen Grundwasserspeicher zeigen beispielhaft, dass einzelne Zonen eines Teilgebietes nicht unabhängig voneinander zum Abfluss am Teilgebietsauslass beitragen. Dieser Abweichung zum Design des HydPy-Frameworks wird Rechnung getragen, indem die HBV_{96}-Zonen als Teilzonen aufgefasst werden. Damit lässt sich im Framework einer einzelnen Zone, die von anderen Zonen des Teilgebietes unabhängig ist, eine Gruppe von Teilzonen vom Typ HBV_{96} zuordnen. So ist HBV_{96} auch innerhalb eines Teilgebietes beliebig mit anderen Modellen kombinierbar.

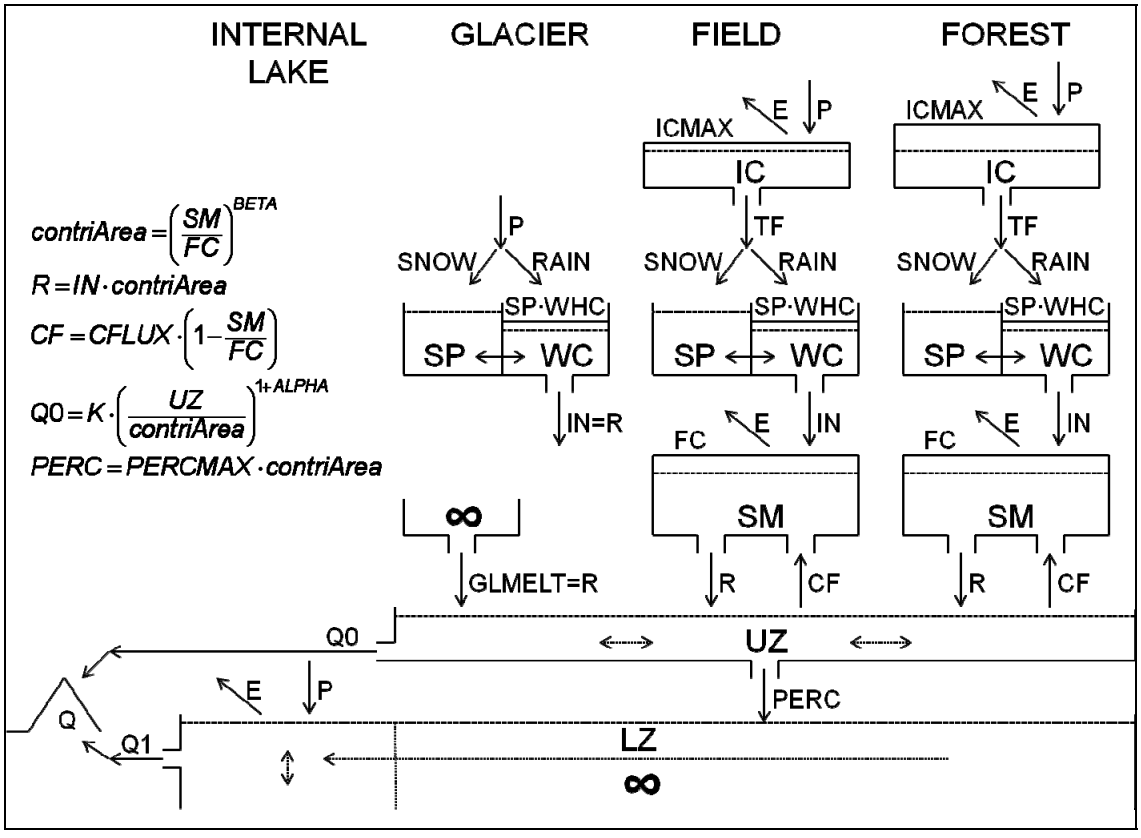


Abbildung 2.1: Schematische Skizze eines Teilgebietes nach HBV_{96} ohne „Outlet Lake“

2.1.1 Meteorologische Randbedingungen

Bis auf Zonen vom Typ „Glacier“, für welche die Verdunstung vernachlässigt wird, benötigt jede HBV_{96}-Zone Zeitreihen des Niederschlages, der Temperatur und der potenziellen Verdunstung. Diese werden zu Beginn eines Modelllaufes aus den für ein gesamtes Teilgebiet gültigen Zeitreihen und Normwerten abgeleitet. Die hierbei zum Tragen kommenden Parameter sind modifizierbar, was die Einbeziehung der Eingangsdaten-Aufbereitung in die Modellkalibrierung ermöglicht.

Die Temperatur wird für jede Zone entsprechend dem Gradienten *TCALT* und dem Bezugsniveau *zrelT* höhenkorrigiert (Gleichung 2.1).

$$Tc_i^k = T_i - TCALT \cdot \left(z^k - zrelT \right) \qquad 2.1$$

mit:	i	Zeitindex[1]	[-]
	k	Zonenindex[2]	[-]
	Tc	Temperatur der Zone	[°C]
	T	gegebene Temperatur	[°C]
	TCALT	Temperaturgradient	[°C/100 m]
	z	Höhe der Zone über NN	[100 m]
	zrelT	Referenzhöhe über NN	[100 m]

Für die Berechnung der Verdunstung (s.u.) erfolgt eine flächengewichtete Mittelung der für die Zonen berechneten Temperaturwerte (Gleichung 2.2).

$$Tmean_i = \sum_{k \in Z} \left(\frac{A^k}{\sum_{k \in Z} A^k} \cdot Tc_i^k \right) \qquad 2.2$$

mit:	Tmean	mittlere Temperatur Teilgebiet	[°C]
	A	Fläche einer Zone	[m²]
	Z	Indexmenge aller Zonen eines Teilgebie-	[m²]

Der flüssige Anteil des nicht korrigierten Niederschlages *fracRain* wird im Temperaturintervall *TTINT* um den Temperaturgrenzwert *TT* linear interpoliert (Gleichung 2.3).

$$fracRain_i^k = \min\left(\max\left(\frac{Tc_i^k - \left(TT - TTINT/2 \right)}{TTINT}; 0 \right); 1 \right) \qquad 2.3$$

mit:	fracRain	flüssiger Niederschlagsanteil	[-]
	TT	Grenzwert flüssiger/fester Niederschlag	[°C]
	TTINT	Intervall mit Mischniederschlag	[°C]

Die Manipulation der Niederschlagshöhe eines Teilgebietes ist global über *PCORR* möglich (Gleichung 2.4). Davon wird in der BfG-Implementierung nur Gebrauch gemacht, um gegebene Eingangsreihen in die Einheit mm/h zu überführen.

---

[1] Bei Flussgrößen bezieht sich i=1 auf das erste Zeitschrittintervall. Bei den weiter unten eingeführten Speichergrößen bezieht sich i=1 auf den Start und j=1 auf das Ende des ersten Zeitintervalls.

[2] Enthält eine Größe keinen Zonenindex, ist diese für ein gesamtes Teilgebiet konstant.

$$Pc_i = PCORR \cdot P_i \tag{2.4}$$

mit:	Pc	manipulierter Niederschlag Teilgebiet	[mm/h]
	PCORR	allgemeiner Niederschlagsfaktor	[-]
	P	gegebener Niederschlag	[mm/h]

Zusätzlich können eine höhenabhängige und eine von der Niederschlagsform abhängige Korrektur in Abhängigkeit des Höhengradienten *PCALT* und der Faktoren *RFCF* und *SFCF* durchgeführt werden (Gleichung 2.5). Es ist zu beachten, dass durch die Wahl verschiedener Werte für *RFCF* und *SFCF* der Wert von *fracRain* nicht mehr dem Anteil des flüssigen Niederschlages entspricht.

$$Pc_i^k = \begin{array}{c} Pc_i \cdot \left(1 + PCALT \cdot \left(z^k - zrelP\right)\right) \cdot \ldots \\ \ldots \left(fracRain_i^k \cdot RFCF + \left(1 - fracRain_i^k\right) \cdot SFCF\right) \end{array} \tag{2.5}$$

mit:	Pc	Niederschlag Teilgebiet bzw. Zone	[mm/h]
	PCALT	Niederschlagsgradient	[mm/h/100 m]
	zrelP	Referenzhöhe über NN	[100 m]
	RFCF	Regenkorrekturfaktor	[-]
	SFCF	Schneekorrekturfaktor	[-]

Werte der potenziellen Verdunstung *EP* werden für ein Teilgebiet unter Rückgriff auf Normwerte der potenziellen Verdunstung *EPn* und der Temperatur *Tn* abgeleitet (Gleichung 2.6). Die Abweichung der aktuellen potenziellen Verdunstung von der Normverdunstung ist proportional zur Abweichung der aktuellen Temperatur von der Normtemperatur, falls *ETF* größer Null. Negative Verdunstungswerte werden auf null zurückgesetzt. Daraus resultiert eine obere Grenze der potenziellen Verdunstung von 2·*EPn*, um den langfristigen Mittelwert der potenziellen Verdunstung weitgehend zu erhalten. Für die BfG-Implementierung wurden die Normwerte der potenziellen Verdunstung als mittlere Monatswerte und die der Temperatur als mittlere Tageswerte bestimmt. Somit sind alle berechneten Stundenwerte der potenziellen Verdunstung eines gegebenen Tages identisch.

$$EP_i = \min\left(\max\left(EPn_i \cdot \left(1 + ETF \cdot \left(Tmean_i - Tn_i\right)\right); 0\right); 2 \cdot EPn_i\right) \tag{2.6}$$

mit:	EP	potenzielle Verdunstung Teilgebiet	[mm/h]
	EPn	potenzielle Normverdunstung	[mm/h]
	Tn	potenzielle Normtemperatur	[°C]
	ETF	Temperaturfaktor der Verdunstung	[1/°C]

Die Ermittlung der potenziellen Verdunstung der Zonen erfolgt über den allgemeinen Korrekturfaktor *ECORR* und den Höhengradienten *ECALT*, wobei zusätzlich eine Reduktion in Anhängigkeit der Niederschlagshöhe über *ETF* möglich ist (Gleichung 2.7). *ECALT* ist in der BfG-Implementierung auf 0 gesetzt, womit die potenzielle Verdunstung an niederschlagsfreien Ta-

gen unabhängig vom Höhenniveau der Zonen ist. *ECORR* dient ausschließlich der Überführung der Verdunstungsreihen in die Einheit mm/h.

$$EPc_i^k = EP_i \cdot ECORR \cdot \left(1 - ECALT \cdot \left(z^k - zrelE\right)\right) \cdot \exp\left(-EPF \cdot Pc_i^k\right) \qquad 2.7$$

mit:	EPc	korrigierte potenzielle Verdunstung Zone	[mm/h]
	ECORR	allgemeiner Verdunstungsfaktor	[-]
	ECALT	Verdunstungsgradient	[mm/h/100 m]
	zrelE	Referenzhöhe über NN	[100 m]
	EPF	Korrekturfaktor bzgl. Niederschlag	[h/mm]

2.1.2 Interzeption

In Zonen vom Typ „Field“ und „Forest“ können Interzeptionsprozesse berücksichtigt werden. Eine Unterscheidung zwischen flüssigem und festem Niederschlagsanteil erfolgt hierbei nicht.

Niederschlag passiert die Pflanzendecke (throughfall *TF*) bei Überschreitung derer maximalen Interzeptionsspeicherung *ICMAX* (Gleichung 2.8). Zur Berücksichtigung der unterschiedlichen Vegetationstypen werden in der BfG-Implementierung Zonen der Typen „Field“ und „Forest“ unterschiedliche Werte für *ICMAX* innerhalb eines Teilgebietes zugewiesen.

$$TF_i^k = \max\left(Pc_i^k - \frac{IC_i^k - ICMAX^k}{dt};\ 0\right) \qquad 2.8$$

mit:	TF	Throughfall	[mm/h]
	IC	Interzeptionsspeicher	[mm]
	ICMAX	maximale Interzeptionsspeicherung	[mm]
	dt	Zeitschrittweite	[h]

Vor Berechnung der Interzeptionsverdunstung *EI* wird der Speicherinhalt bezüglich *Pc* und *TF* aktualisiert (Gleichung 2.9).

$$IC_t^k = IC_i^k + dt \cdot \left(Pc_i^k - TF_i^k\right) \qquad 2.9$$

mit:	t	Index für temporären Zustand[3]	[mm/h]

[3] Mit temporären Zustandsindizes werden Größen versehen, die aufgrund des sequentiellen Lösungscharakters des HBV_{96}-Modells mehrfach berechnet bzw. aktualisiert werden. Sind mehrere Zwischenschritte notwendig, wird zusätzlich ein laufender Index angehängt.

EI entspricht der potenziellen Verdunstung oder der im Interzeptionsspeicher vorhandenen Wassermenge (Gleichung 2.10).

$$EI_i^k = \min\left(EPc_i^k;\ \frac{IC_t^k}{dt} \right) \qquad 2.10$$

mit: EI Interzeptionsverdunstung [mm/h]

Durch abschließende Subtraktion der Interzeptionsverdunstung wird der Endzustand des Interzeptionsspeichers zum gegebenen Zeitschritt ermittelt (Gleichung 2.11).

$$IC_j^k = IC_t^k - dt \cdot EI_i^k \qquad 2.11$$

mit: j^1 i+1 [mm/h]

Für Zonen vom Typ „Ilake“ und „Glacier“ gilt Gleichung 2.12.

$$TF_i^k = Pc_i^k \qquad 2.12$$

2.1.3 Schnee und Gletscher

Schneeprozesse sind lediglich für die Zonentypen „Field“, „Forest“ und „Glacier“ vorgesehen. Im Gegensatz zu Vorgängerversionen von HBV_{96} wird die Schneedecke über zwei Speichergrößen abgebildet: *SP* repräsentiert deren Eisanteil, *WC* deren Wasseranteil. Verdunstung aus der Schneedecke findet nicht statt. Keine Berücksichtigung findet die Sammlung verschiedener Schneemengen innerhalb einer Zone in „Schneeklassen“, da in der BfG-Implementierung hiervon nicht Gebrauch gemacht wird.

Zunächst wird der, für die Zonentypen „Field“ und „Forest“ den Interzeptionsspeicher passierende, aktuelle Niederschlag anteilig zum Wasser- (Gleichung 2.13) und zum Eisgehalt (Gleichung 2.14) der Schneedecke addiert. Die relativen Anteile des flüssigen und festen Niederschlags werden durch die Interzeptionsprozesse nicht verändert.

$$WC_{t_1}^k = WC_i^k + dt \cdot TF_i^k \cdot \frac{fracRain_i^k \cdot RFCF}{fracRain_i^k \cdot RFCF + \left(1 - fracRain\right)_i^k \cdot SFCF} \qquad 2.13$$

mit: WC Wassergehalt der Schneedecke [mm]

$$SP_t^k = SP_i^k + dt \cdot TF_i^k \cdot \frac{\left(1 - fracRain_i^k\right) \cdot SFCF}{fracRain_i^k \cdot RFCF + \left(1 - fracRain\right)_i^k \cdot SFCF} \qquad 2.14$$

mit: SP Eisgehalt der Schneedecke [mm]

Die Berechnung der Schneeschmelze erfolgt nach dem Taggradfaktor-Ansatz (Gleichung 2.15), wobei der Faktor *CFMAX* für „Forest"-Zonen abweichend festgelegt werden kann, um Beschattungseffekte zu berücksichtigen.

$$MELT_i^k = \min\left(\max\left(CFMAX \cdot \left(Tc_i^k - TTM \right); 0 \right); \frac{SP_t^k}{dt} \right) \qquad 2.15$$

mit:	MELT	Schmelze des Eisgehaltes	[mm/h]
	CFMAX	Taggradfaktor Schneedecke	[mm/h/°C]
	TTM	Schmelzpunkt	[°C]

Der Wasseranteil der Schneedecke gefriert unterhalb des Schmelzpunktes *TTM*, falls *CFR* größer Null (Gleichung 2.16).

$$REFR_i^k = \min\left(\max\left(CFR \cdot CFMAX \cdot \left(TTM - Tc_i^k \right); 0 \right); \frac{WC_{t_1}^k}{dt} \right) \qquad 2.16$$

mit:	REFR	(Wieder-)Gefrieren des Wassergehaltes	[mm/h]
	CFR	Gefrierfaktor	[-]

Dem Wassergehalt wird das Schmelzwasser auf- und das gefrierende Wasser abgeschlagen (Gleichung 2.17), dem Eisanteil das gefrierende Wasser zugezählt und das Schmelzwasser abgezogen (Gleichung 2.18).

$$WC_{t_2}^k = WC_{t_1}^k + dt \cdot \left(MELT_i^k - REFR_i^k \right) \qquad 2.17$$

$$SP_j^k = SP_t^k + dt \cdot \left(REFR_i^k - MELT_i^k \right) \qquad 2.18$$

Die die Haltekapazität der Schneedecke überschreitende Wassermenge wird freigesetzt (Gleichung 2.19). Die absolute Wasserhaltekapazität ist proportional zum absoluten Eisgehalt des Schnees.

$$IN_t^k = \max\left(\frac{WC_{t_2}^k - \left(WHC \cdot SP_j^k \right)}{dt}; 0 \right) \qquad 2.19$$

mit:	IN	freigesetztes Wasser	[mm/h]
	WHC	relative Wasserhaltekapazität	[-]

Der finale Wassergehalt der Schneedecke ergibt sich durch Subtraktion des freigesetzten Wassers (Gleichung 2.20).

$$WC_j^k = WC_{t_2}^k - dt \cdot IN_t^k \qquad 2.20$$

In „Forest"- und „Field"-Zonen wird nur das aus der Schneedecke freigesetzte bzw. das nicht von der Schneedecke gehaltene Wasser an das Bodenmodul weitergereicht (Gleichung 2.21).

$$IN_i^k = IN_t^k \qquad 2.21$$

In „Glacier"-Zonen wird, falls zum Zeitschrittende keine Schneedecke vorhanden ist, ggf. Gletscherschmelzwasser hinzuaddiert (Gleichung 2.22). Hierfür kommt der von *CFMAX* unabhängige Taggradfaktor *GMELT* zum Einsatz. Eine Speisung des Gletschers durch Niederschlag erfolgt nicht, womit ein Gletscher eine zusätzliche, unendliche Wasserquelle darstellt.

$$IN_i^k = \begin{cases} IN_t^k + GMELT \cdot \left(Tc_i^k - TTM\right) & falls \quad \left(SP_j^k = 0\right) \cap \left(Tc_i^k > TTM\right) \\ IN_t^k & falls \quad \left(SP_j^k > 0\right) \cup \left(Tc_i^k \le TTM\right) \end{cases} \qquad 2.22$$

mit: GMELT Taggradfaktor Gletscher [mm/h/°C]

In „Internal Lake"-Zonen werden Schneeprozesse vernachlässigt, weshalb Gleichung 2.23 gilt.

$$IN_i^k = TF_i^k \qquad 2.23$$

2.1.4 Boden

Der Bodenspeicher ist das zentrale Element in der Berechnung des Effektivniederschlages und der Verdunstung der „Field"- und „Forest"-Zonen. Für Zonen vom Typ „Glacier" ist kein Bodenmodul implementiert, da die Verdunstung bei diesen vernachlässigt wird und sich somit – entsprechend der Bodenspeicherkonzeption – zwangsläufig ein Abflussbeiwert von eins ergibt.

Das Verhältnis aus potenziellem Abfluss[4] *R* und den Boden erreichendem Niederschlags- und Schmelzwasser steht (falls *BETA* ungleich eins ist) in einer nichtlinearer Abhängigkeit zur relativen Bodenfeuchte (Gleichung 2.24).

[4] Sind keine „Internal Lake"-Zonen in einem Teilgebiet vorhanden und ist der maximale kapillare Wiederaufstieg gleich Null, kann *R* als Effektivniederschlag bzw. noch nicht zeitlich verzögerter Abfluss aufgefasst werden. Anderenfalls ist *R* ggf. der Verdunstung zugänglich, deshalb die Formulierung „potenzieller Abfluss".

$$R_i^k = IN_i^k \cdot \left(\frac{SM_i^k + 0{,}5 \cdot dt \cdot IN_i^k}{FC} \right)^{BETA} \qquad 2.24$$

mit:			
	R	(potenzieller) Abfluss	[mm/h]
	SM	momentane Bodenspeicherfüllung	[mm]
	FC	maximale Bodenspeicherfüllung	[mm]
	BETA	Exponent Bodenspeicher	[-]

Der nicht potenziell abflusswirksame Anteil von *IN* wird vor anschließenden Wasserhaushaltsberechnungen *SM* aufgeschlagen (Gleichung 2.25).

$$SM_{t_1}^k = SM_i^k + dt \cdot \left(IN_i^k - R_i^k \right) \qquad 2.25$$

Abhängig vom relativen Fülldefizit des Bodenspeichers beträgt der kapillare Aufstieg *CF* aus dem Direktabflussspeicher bzw. dem oberen Grundwasserspeicher *UZ* (siehe Diskussion unten) in den Bodenspeicher maximal *CFLUX* (Gleichung 2.26). Es ist zu beachten, dass *SM* und *UZ* i. d. R. unterschiedliche Flächengrößen aufweisen, da *SM* einer Zone und *UZ* einem Teilgebiet (abzüglich „Internal Lake"-Zonen) zugeordnet ist. Für den kapillaren Aufstieg in den Bodenspeicher einer Zone steht aber nur der Anteil des Wassers in *UZ* zur Verfügung, welcher dem Flächenanteil der Zone an der Gesamtfläche aller Zonen im Teilgebiet (abzüglich „Internal Lake"-Zonen) entspricht. Anschaulicher formuliert kann nur Wasser aus dem oberen Grundwasserspeicher in eine Bodenzone aufsteigen, welches sich „direkt unter dieser befindet".

$$CF_i^k = \min\left(CFLUX \cdot \left(1 - \frac{SM_{t_1}^k}{FC} \right); \frac{UZ_i}{dt} + R_i^k \right) \qquad 2.26$$

mit:			
	CF	momentaner kapillarer Aufstieg	[mm/h]
	CFLUX	maximaler kapillarer Aufstieg	[mm/h]
	UZ	Direktabflussspeicher	[mm]

Vor der Berechnung der Verdunstung aus dem Bodenspeicher wird der kapillare Aufstieg hinzuaddiert (Gleichung 2.27).

$$SM_{t_2}^k = SM_{t_1}^k + dt \cdot CF_i^k \qquad 2.27$$

Verdunstung aus dem Bodenspeicher *EA* tritt nur auf, wenn keine Schneedecke vorhanden ist, d. h. wenn der Speicher für den Eisgehalt der Schneedecke zum Zeitschrittende leer ist (Gleichung 2.28). Aktuelle und potenzielle Verdunstung entsprechen einander bei Bodenspeichergehalten größer oder gleich *LP·FC*.

$$EA_t^k = \begin{cases} \min\left(EPc_i^k \cdot \frac{SM_{t_2}^k}{LP \cdot FC}; EPc_i^k; \frac{SM_{t_2}^k}{dt} \right) & falls \quad SP_j^k = 0 \\ 0 & falls \quad SP_j^k > 0 \end{cases} \qquad 2.28$$

mit: EA Verdunstung aus dem Bodenspeicher [mm/h]

LP Verdunstungsparameter [-]

Durch die unabhängige Berechnung von Interzeptions- und Bodenverdunstung kann deren Summe bis zum doppelten der potenziellen Verdunstung betragen. Die Gesamtverdunstung oberhalb der potenziellen Verdunstung wird entsprechend dem Reduktionsparameter *ERED* anteilig verringert (Gleichung 2.29).

$$EA_i^k = EA_t^k - \max\left(ERED \cdot \left(EA_t^k + EI_i^k - EPc_i^k \right); 0 \right) \qquad 2.29$$

mit: ERED Reduktion überpotenzieller Verdunstung [-]

Durch Subtraktion der tatsächlichen Verdunstung wird der finale Bodenspeicherinhalt eines Zeitschrittes bestimmt (Gleichung 2.30).

$$SM_j^k = SM_{t_2}^k - dt \cdot EA_i^k \qquad 2.30$$

Für „Glacier“ und „Internal Lake“-Zonen gilt Gleichung 2.31:

$$R_i^k = IN_i^k \qquad 2.31$$

2.1.5 Abflusskonzentration bzw. Grundwasser

Der „untere“ Bereich der HBV_{96}-Konzeption hat eine Doppelfunktion. Einerseits dienen die Speicher *UZ* (upper zone layer) und *LZ* (lower zone layer) der Umsetzung von Abflusskonzentrationsprozessen: *UZ* bewirkt die zeitliche Verzögerung der schnellen Abflusskomponente; *LZ* die der langsamen. Andererseits lassen sich die Speicher dem Grundwasser zuordnen: UZ dem oberflächennahem Grundwasser, aus dem es zu kapillarem Wiederaufstieg in die Bodenzone und Durchsickerung ins tiefere Grundwasser kommt; LZ entspricht dem tiefem Grundwasser, das Seen speist und unabhängig vom Niederschlag dauerhaft existent ist. Jedes Teilgebiet enthält lediglich einen „upper zone-“ und einen „lower zone-layer“, wodurch es zu indirekten Wechselwirkungen der ansonsten voneinander unabhängigen Zonen im Teilgebiet kommt.

In *UZ* entwässern alle Zonen der Typen „Field“, „Forest“ und „Glacier“. Daher wird *UZ* im ersten Schritt entsprechend dem potenziellen Abfluss *R* und ggf. dem kapillaren Aufstieg *CF* flächengewichtet aktualisiert (Gleichung 2.32).

$$UZ_{t_1} = UZ_i + dt \cdot \sum_{k \in Z_L} \left(\frac{A^k}{\sum_{k \in Z_L} A^k} \cdot \left(R_i^k - CF_i^k \right) \right) \qquad 2.32$$

mit: Z_L Indexmenge aller „Land“-Zonen [-]

Um die Abhängigkeit der Hochwasser-Rezessionskurve von der Gebietsfeuchte abzubilden, wurde nach der Ersteinführung von HBV_{96} ein contributing area-Ansatz in die Berechnung der Ausflüsse des Speichers UZ implementiert (SMHI, 2009). Dieser wird über den Parameter *RESPAREA* aktiviert. Zur Berechnung des zum Abfluss beitragenden Flächenanteils *contriArea* wird auf den relativen Bodenwassergehalt aller Zonen vom Typ „Field“ und „Forest“ zurückgegriffen. Sind keine Zonen dieser Typen im Teilgebiet enthalten, ist *contriArea* gleich eins, was der HBV_{96}-Umsetzung ohne contributing area-Ansatz entspricht (Gleichung 2.33).

$$contriArea_i = \begin{cases} \sum_{k \in Z_S} \left(\frac{A^k}{\sum_{k \in Z_S} A^k} \cdot \left(\frac{SM_j^k}{FC} \right)^{BETA} \right) & falls \quad (RESPAREA = 1) \cap (Z_S \neq \varnothing) \\ 1 & falls \quad (RESPAREA = 0) \cup (Z_S = \varnothing) \end{cases} \qquad 2.33$$

mit: contriArea zum Abfluss beitragender Flächenanteil [-]
Z_L Indexmenge aller „Soil“-Zonen [-]

Sofern ausreichend Wasser in *UZ* vorhanden ist, perkoliert das oberflächennahe in das tiefe Grundwasser entsprechend dem gesättigten Flächenanteil und der maximal möglichen Perkolation *PERCMAX* (Gleichung 2.34).

$$PERC_i = \min \left(PERCMAX \cdot contriArea_i ; \frac{UZ_{t_1}}{dt} \right) \qquad 2.34$$

mit: PERC momentane Perkolation [mm/h]
PERCMAX maximal mögliche Perkolation [mm/h]

Vor Berechnung der schnellen Abflusskomponente erfolgt eine Aktualisierung von *UZ* (Glei-

chung 2.35).

$$UZ_{t_2} = UZ_{t_1} - dt \cdot PERC_i \qquad 2.35$$

Um den steileren Abflussrückgang sommerlicher Hochwasser bei eher geringer Gebietsfeuchte umzusetzen, wird *UZ* in der Berechnung der schnellen Abflusskomponente *Q0* durch *contriArea* dividiert (Gleichung 2.36).

$$Q0_i = \min\left(K \cdot \left(\frac{UZ_{t_2}}{contriArea_i} \right)^{1+ALPHA} ; \frac{UZ_{t_2}}{dt} \right) \qquad 2.36$$

mit:	Q0	schnelle Abflusskomponente	[mm/h]
	K	Rückgangskoeffizient	[1/h/mm^{ALPHA}]
	ALPHA	Exponent	[-]

Die Speicherkonstante *K* wird vorab aus den Parametern *HQ* und *KHQ* ermittelt. Für einen Hochwasserdurchfluss *HQ* kann der Rückgangskoeffizient *KHQ* aus gemessenen Ganglinien z. B. graphisch ermittelt werden. *K* wird nach Gleichung 2.37 so festgelegt, dass sowohl ein linearer Einzelspeicher mit *KHQ* als auch der entsprechend *ALPHA* nichtlinearer Speicher *UZ* mit *K* bei identischem Füllungsstand zu *HQ* führen.

$$K = \frac{HQ}{\left(HQ/KHQ\right)^{ALPHA+1}} \qquad 2.37$$

mit:	HQ	Bezugshochwasser	[mm/h]
	KHQ	Linearspeicherkonstante für HQ	[1/h]

Abschließend wird *Q0* von *UZ* subtrahiert (Gleichung 2.38).

$$UZ_j = UZ_{t_2} - dt \cdot Q0_i \qquad 2.38$$

Aufgrund der i. d. R. großen zeitlichen relativen Variabilität von *UZ* sind in dieser Modellkomponente die größten numerischen Fehler durch den sequenziellen Lösungsweg von HBV$_{96}$ zu erwarten. Diese lassen sich reduzieren durch Belegung des Parameters *RECSTEP* mit Werten, die größer als die Anzahl der Rechenschritte pro Tag sind. Bei einem *RECSTEP*-Wert von 48 bei einer ansonsten gültigen Rechenschrittweite von einer Stunde werden die Gleichungen 2.32 sowie 2.34 bis 2.38 mit einem Zeitschritt *dt* von 0,5 h zweimal durchlaufen, wobei zusätzlich eine Akkumulation der Flussgrößen für den weiteren Berechnungsgang erfolgt. Von einer automatisierten und dynamischen Ermittlung des *RECSTEP*-Wertes, wie in der HBV-BfG-Implementierung genutzt, wurde abgesehen. Die Wahl eines konstanten, hohen Wertes für *RECSTEP* erwies sich als zielführender (siehe 2.1.8).

Dem tiefen Grundwasserspeicher *LZ* eines Teilgebietes wird das aus *UZ* perkolierende Wasser aufgeschlagen. Sind „Internal Lake“-Zonen im Teilgebiet vorhanden, wird zusätzlich deren potenzieller Abfluss *R*, d. h. der die Seeoberfläche erreichende Zonenniederschlag *Pc* hinzugerechnet. Dabei wird entsprechend den Flächengrößen der jeweiligen Zonen gewichtet (Glei-

chung 2.39).

$$LZ_{t_1} = LZ_i + \frac{\sum_{k \in Z_L} A^k}{\sum_{k \in Z} A^k} \cdot PERC_i + \sum_{k \in Z_I} \left(\frac{A^k}{\sum_{k \in Z} A^k} \cdot R_i^k \right) \qquad 2.39$$

mit:	LZ	Basisabflussspeicher	[mm]
	Z_I	Indexmenge aller „Internal Lake“-Zonen	[-]

Vorausgesetzt die Temperatur überschreitet der Grenztemperatur *TTICE*, verdunstet mit der potenziellen Verdunstungsrate aus dem von „Internal Lake“-Zonen eingenommenen Flächenanteil des Teilgebietes tiefes Grund- bzw. Seewasser (Gleichung 2.40). Eine Beschränkung entsprechend dem Füllstand von *LZ* erfolgt nicht, wodurch dieser negative Werte annehmen kann.

$$LZ_{t_2} = LZ_{t_1} - \sum_{k \in Z_I | Tc_i^k > TTICE} \left(\frac{A^k}{\sum_{k \in Z} A^k} \cdot EPc_i^k \right) \qquad 2.40$$

mit:	TTICE	Grenztemperatur Seeeis	[°C]

Die langsame Abflusskomponente *Q1* berechnet sich nach der Einzellinearspeichergleichung, falls *LZ* positive Werte annimmt (Gleichung 2.41).

$$Q1_i = \max\left(K4 \cdot LZ_{t_2}; 0\right) \qquad 2.41$$

mit:	Q1	langsame Abflusskomponente	[mm/h]
	K4	Linearspeicherkonstante	[1/h]

Abschließend wird *LZ* um die langsame Abflusskomponente aktualisiert (Gleichung 2.42).

$$LZ_j = LZ_{t_2} - dt \cdot Q1_i \qquad 2.42$$

Zusätzlich zur oben beschriebenen Retention des Abflusses wird mit einem Dreiecks-Unit-Hydrograph eine weitere zeitliche Verzögerung des Abflusses bewirkt. Eine Ordinate berechnen sich nach Gleichung 2.43, wobei *MAXBAZ* die Basislänge des gleichschenkligen Dreiecks darstellt.

$$UH_i = 4 \cdot \int_{i-1}^{i} \max\left(\min\left(MAXBAZ^{-2} \cdot i; MAXBAZ^{-1} - MAXBAZ^{-2} \cdot i\right); 0\right) di \qquad 2.43$$

mit:	UH	Dreiecks-Unit-Hydrograph-Ordinate	[-]
	MAXBAZ	Basislänge gleichschenkliges Dreieck	[h]

Der Abflusswert eines Zeitschrittes aller Zonen eines Teilgebietes am Gebietsauslass oder ggf. am Zulauf einer „Outlet Lake“-Zone ergibt sich durch das Skalarprodukt des Vektors der Unit-Hydrograph-Ordinaten mit dem Vektor der n_{UH} vorangegangenen Summen des Basisabflusses und des Direktabflusses der „Land“-Zonen (Gleichung 2.44).

$$Q_{i,t} = \overrightarrow{UH} \cdot \left(\frac{\sum_{k \in Z_L} A^k}{\sum_{k \in Z} A^k} \cdot \overrightarrow{Q0}_{i:i-n_{UH}} + \overrightarrow{Q1}_{i:i-n_{UH}} \right) \qquad 2.44$$

mit:	Q	Zonen-Gesamtabfluss am Gebietsauslass	[mm/h]
	n_{UH}	Anzahl der Unit-Hydrograph-Ordinaten	[-]

Über den Parameter *ABSTR* lässt sich der berechnete Teilgebiets-Abfluss der Zonen *Q* um einen konstanten Wert erhöhen oder – unter Berücksichtigung der Nichtnegativität – verringern (Gleichung 2.45).

$$Q_i = \max\left(Q_{i,t} - ABSTR; 0\right) \qquad 2.45$$

2.1.6 Seen und Talsperren

Da keine „outlet lakes“ in der HBV-BfG-Implementierung berücksichtigt sind, wurden diese bislang nicht umgesetzt.

2.1.7 Wellenablauf

In HBV_{96}-IHMS gilt die Annahme, dass ein Teilgebiet für jedes der maximal fünf Oberlieger-Gebiete einen separaten Flusslauf enthält. Somit nimmt jeder Flusslauf im Modell den Zufluss nur eines Teilgebietes auf. Am Gebietsauslass werden alle Ausflüsse der Flussläufe sowie der Zonen des Teilgebietes aufsummiert. Die „nachträgliche“ Zusammenführung der Zuflüsse der Oberlieger-Gebiete führt, eine identische Parametrisierung vorausgesetzt, aufgrund der Linearität der Methode zur Wellenablauf-Berechnung zum selben Ergebnis wie die unmittelbare Zusammenführung und gemeinsame Abführung durch einen Flusslauf.

Aus den beschriebenen modelltechnischen Randbedingungen resultiert eine Besonderheit der IHMS-Konfiguration des Rhein-Flussgebietes: zum Teil enthalten die HBV-Teilgebiete reale Einzugsgebiete, die nicht direkt über Flussläufe verbunden sind. Ein Beispiel ist das Teilgebiet „WeschMod“, welches aus einem großen Teil des Einzugsgebietes der Weschnitz und einem kleinen Kopfgebiet der nördlich gelegenen Modau besteht. Beide entwässern erst einige Kilometer stromab in den Rhein. Zwei realen Gebietsauslässen steht damit ein Modell-Gebietsauslass entgegen. Ein Vergleich von Simulationsrechnungen mit Messungen ist daher nur durch Addition der Abflussreihen der an den beiden realen Gebietsauslässen befindlichen Pegel Lorsch und Eberstadt möglich. In der HydPy-Implementierung werden gemessene Abflussreihen in derartigen Fällen bereits addiert vorgehalten.

Des Weiteren wird in der IHMS-Implementierung von der Bifurkations-Option Gebrauch gemacht. Der Ausfluss eines Teilgebietes kann auf zwei unterhalb gelegene Teilgebiete aufgeteilt

werden. Die Aufteilung erfolgt von der Durchflussmenge abhängig anhand von tabellarisch gegebenen Stützstellen, zwischen denen linear interpoliert wird. Außerhalb des tabellierten Durchflussbereiches wird anhand der beiden höchsten Stützstellen linear extrapoliert. Die Bifurkation erfolgt im Rhein-Modell an mehreren Stellen mit zwei Zielen: erstens, um in verschiedenen Abflussbereichen verschiedene Routing-Parameter anzuwenden und zweitens, um bei großen Hochwassern Abflussanteile aus dem System zu entfernen. Hierfür werden künstliche Teilgebiete angelegt und nur Teile des Gesamtdurchflusses in die eigentlichen Teilgebiete zurückgeführt. In der HydPy-Implementierung erübrigt sich das Anlegen künstlicher Teilgebiete, da jede Flusslauf-Instanz – falls gegeben – aufgrund der in den Parametervektoren *Qtotal* und *Qbranch* enthaltenen Stützstellen nur den gewünschten Teil des Ausflusses des ihm zugeordneten Oberlieger-Gebietes aufnimmt.

Zur Wellenablaufberechnung kann jeder Flusslauf entsprechend dem Translations-Parameter *LAG* in mehrere Abschnitte unterteilt werden. Ist, wie üblich, *LAG* auf die Tagesschrittweite bezogen und soll die Simulation in der Stundenschrittweite erfolgen, entspricht ein *LAG*-Wert von eins vierundzwanzig Gerinneabschnitten. Die Berechnung des Ausflusses *Q* aus dem Gerinneabschnitt *x* zum Zeitindex *i* erfolgt mit der Arbeitsgleichung des Muskingum-Verfahrens (Gleichung 2.46).

$$Q_i^x = c_1 \cdot Q_i^{x-1} + c_2 \cdot Q_{i-1}^{x-1} + c_3 \cdot Q_{i-1}^x \qquad 2.46$$

mit:	Q	Abfluss	[m³/s]
	i	Zeitschritt-Index	[-]
	x	Gerinneabschnitt-Index	[-]
	c_1, c_2, c_3	Koeffizienten	[-]

Die Koeffizienten c_1, c_2 und c_3 berechnen sich aus dem Retentions-Parameter *DAMP* (Gleichung 2.47).

$$c_1 = c_3 = \frac{DAMP}{1 + DAMP}$$
$$c_2 = 1 - 2 \cdot \frac{DAMP}{1 + DAMP} \qquad 2.47$$

mit:	DAMP	Retentions-Parameter	[-]

LAG beschreibt die zeitliche Verschiebung, beispielsweise einer Hochwasserwelle, *DAMP* deren Abflachung. Ist *LAG* gleich null, sind – unabhängig von *DAMP* – Zu- und Abfluss eines Flusslaufes identisch. Ist *LAG* gleich eins und *DAMP* gleich null, kommt es zu einer zeitlichen Verschiebung von 24 Stunden ohne Formveränderung. Je größer der Wert von *DAMP* im Intervall zwischen null und eins bei konstantem *LAG* gewählt wird, desto größer ist die Abflachung der Welle bei gleichbleibender Schwerpunktlaufzeit.

2.1.8 Vergleich der IHMS- und der HydPy-Implementierung

Zunächst wird die Funktionalität der HBV_{96}-Umsetzung in HydPy für die Zonen der Typen „Field“, „Forest“, „Glacier“ und „Internal Lake“ anhand von vier, sowohl naturräumlich als auch in ihrer modelltechnischen Umsetzung unterschiedlichen Kopfgebieten demonstriert: Thuner See, Neckar 1, Lippe und Emscher. Es erfolgen Vergleiche mit gemessenen Abflüssen sowie mit Simulationsergebnissen des in Stundenschritten rechnenden IHMS-Vorhersagemodells der BfG. Der Vergleichszeitraum umfasst die neun hydrologischen Jahre 1997 bis 2005. Als Anfangsbedingung dienen die für den 01.11.1996 seitens der BfG zur Verfügung gestellten Speicher- und Abflussgrößen.

Teilgebiet Neckar 1

Das Teilgebiet Neckar 1 enthält 7 „Field“- und 6 „Forest“-Zonen die in 100 m-Höhenstufen eingeteilt sind. Die Höhenkorrektur der potenziellen Verdunstung (Gleichung 2.7), die Reduktion der überpotenziellen Verdunstung (Gleichung 2.29) und der kapillaren Wiederaufstieg (Gleichung 2.26) sind nicht umgesetzt, d. h. *ECALT*, *ERED* sowie *CFLUX* sind auf null gesetzt. Vom contributing area-Ansatz (Gleichung 2.33) sowie von der niederschlagsbezogenen Verdunstungskorrektur (Gleichung 2.7) wird dagegen Gebrauch gemacht, d. h. *RESPAREA* ist auf eins und *EPF* ist größer eins gesetzt. Der mittlere gemessene Durchfluss beträgt 16,07 m³/s, die mittleren simulierten Durchflüsse beider HBV_{96}-Umsetzungen 16,80 m³/s. Unterschiede zwischen beiden HBV_{96}-Umsetzungen zeigen sich in der Residuenanalyse nach Gleichung 2.48, verschiedene Quantile der Residuen sind in Tabelle 2.1 aufgeführt.

$$\vec{\varepsilon}_{abs} = \vec{Q}_{HWU} - \vec{Q}_{SMHI}$$
$$\vec{\varepsilon}_{rel} = \frac{\vec{Q}_{HWU} - \vec{Q}_{SMHI}}{\vec{Q}_{SMHI}} \qquad 2.48$$

mit:	ε_{abs}	absolute Abweichung	[m³/s]
	ε_{rel}	relative Abweichung	[-]
	Q_{HydPy}	mit HydPy simulierter Abfluss	[m³/s]
	Q_{IHMS}	mit IHMS simulierter Abfluss	[m³/s]

Tabelle 2.1: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Neckar 1

	Minimum	Quantile							Maximum
		0.01	0.05	0.1	0.5	0.9	0.95	0.99	
ε [m³/s]	-1.22	-0.26	-0.12	-0.09	-0.01	0.06	0.17	0.65	4.16
ε [%]	-1.58	-0.75	-0.66	-0.56	-0.13	0.33	0.59	1.14	6.65

Die maximalen Abweichungen in neun Jahren betragen 4,16 m³/s und 6,65 %, die Mittelwerte der Beträge der Abweichungen 0,07 m³/s und 0,30 %. Die entsprechenden Werte der Abwei-

chungen zwischen gemessenem und mit der IHMS-Implementierung simuliertem Abfluss sind mit 117,82 m³/s und 667,87 % für die Maxima sowie 0,72 m³/s und 23,72 % für die Mittelwerte ungleich größer. Abbildung 2.2 zeigt den gemessenen sowie die simulierten Abflüsse für den Dezember 1997. In diesem Monat tritt die größte relative Abweichung zwischen beiden Modell-Umsetzungen auf. Dennoch sind diese im Vergleich zum Unterschied zum gemessenen Abfluss vernachlässigbar.

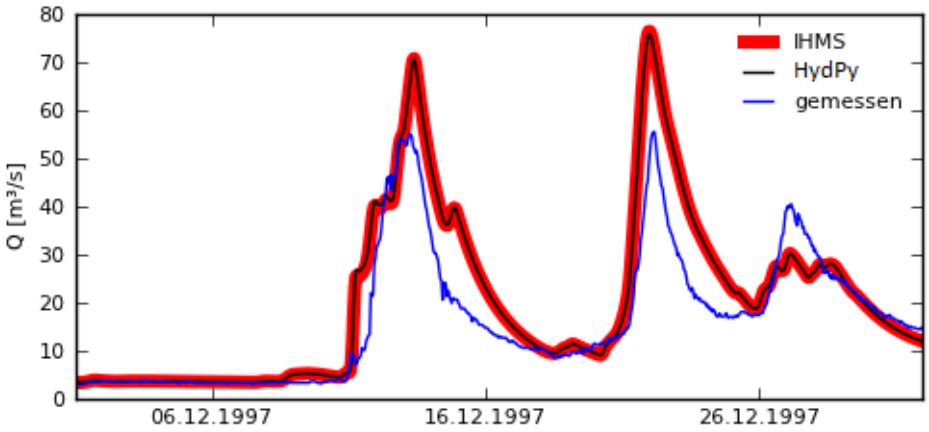


Abbildung 2.2: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Neckar 1

Teilgebiet Thuner See

Im Teilgebiet Thuner See nehmen „Field"- und „Forest"-Zonen eine im Vergleich zu anderen Kopfgebieten geringe Fläche ein. Von 49 Zonen sind 6 vom Typ „Glacier" mit insgesamt 8,6 % Flächenanteil. Allerdings ist der Parameter *GMELT* mit dem Wert null belegt, womit keine Gletscherschmelze erfolgt (Gleichung 2.22). Alle Seen sind in einer Zone mit 4,3 % Flächenanteil zusammengefasst. Die Bildung von Seeeis und die resultierende Verhinderung der Seeverdunstung sind nicht berücksichtigt (Gleichung 2.40). Entgegen dem Teilgebiet Neckar 1 wird die Überschreitung der potenziellen durch die tatsächliche Verdunstung vollständig vermieden (*ERED* gleich eins), wird kein Gebrauch vom contributing area-Ansatz, der niederschlagsbezogenen Verdunstungskorrektur sowie der Niederschlagshöhenkorrektur (Gleichung 2.5) gemacht und wird die potenzielle Verdunstung der „Forest"- gegenüber den „Field"-Zonen 20 % höher angesetzt. Eine gemessene Abflussreihe liegt dem Bearbeiter nicht vor. Die Mittelwerte der mit IMHS und HydPy simulierten Abflussreihen betragen 62,25 m³/s und 62,20 m³/s. Die Quantile der Residuen sind in Tabelle 2.2 gegeben.

Tabelle 2.2: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Thuner See

	Mini-mum	Quantile 0.01	0.05	0.1	0.5	0.9	0.95	0.99	Maxi-mum
ε [m³/s]	-1.00	-0.36	-0.24	-0.18	-0.06	0.05	0.12	0.48	6.94
ε [%]	-0.97	-0.50	-0.34	-0.29	-0.12	0.07	0.17	0.50	3.14

Die maximalen Abweichungen in neun Jahren betragen 6,94 m³/s und 3,14 %, die Mittelwerte der Beträge der Abweichungen 0,11 m³/s und 0,16 %. Abbildung 2.3 zeigt den gemessenen so-

wie die simulierten Abflüsse für den September 1997. Auch wenn kein Vergleich mit gemessenen Abflusswerten möglich ist, erscheint der Unterschied zwischen beiden Modellumsetzungen von geringer Relevanz.

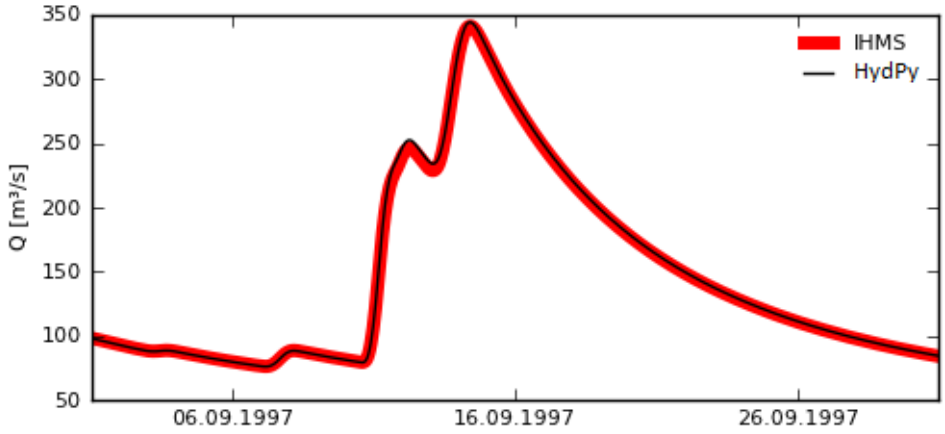


Abbildung 2.3: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Thuner See

Teilgebiet Lippe 1

Das Teilgebiet Lippe 1 enthält 6 „Field"- und 6 „Forest"-Zonen die in 100 m-Höhenstufen eingeteilt sind. Darin und in seiner Parametrisierung ähnelt es stark dem Teilgebiet Neckar 1. Insbesondere wurden dieselben grundsätzlichen Modelloptionen gewählt, wie beispielsweise die Nutzung des contributing area-Ansatzes. Dagegen unterscheidet sich das Teilgebiet Lippe 1 (302 m. ü. NN) naturräumlich durch den Flachlandcharakter der westfälischen Bucht und dem moderaten Eggegebirge vom Teilgebiet Neckar 1 (686 m. ü. NN), das durch Schwarzwald und Schwäbische Alb eher Mittelgebirgscharakter aufweist. Der mittlere gemessene Durchfluss beträgt 17,46 m³/s, die mittleren simulierten Durchflüsse 16,26 m³/s in der IHMS- und 16,25 m³/s in der HydPy-Umsetzung. Die größten positiven und negativen Abweichungen der einzelnen Stundenwerte fallen noch geringer aus als die für Teilgebiet Neckar 1 (Tabelle 2.3). Im Monat mit der stärksten relativen Abweichung zwischen den Modell-Implementierungen, Oktober 2003, sind die Abweichungen zur gemessenen Ganglinie um mehrere Ordnungen größer.

Tabelle 2.3: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Lippe 1

	Minimum	Quantile							Maximum
		0.01	0.05	0.1	0.5	0.9	0.95	0.99	
ε [m³/s]	-0.18	-0.14	-0.10	-0.08	-0.01	0.03	0.11	0.40	1.14
ε [%]	-0.71	-0.63	-0.57	-0.47	-0.11	0.20	0.40	0.93	1.57

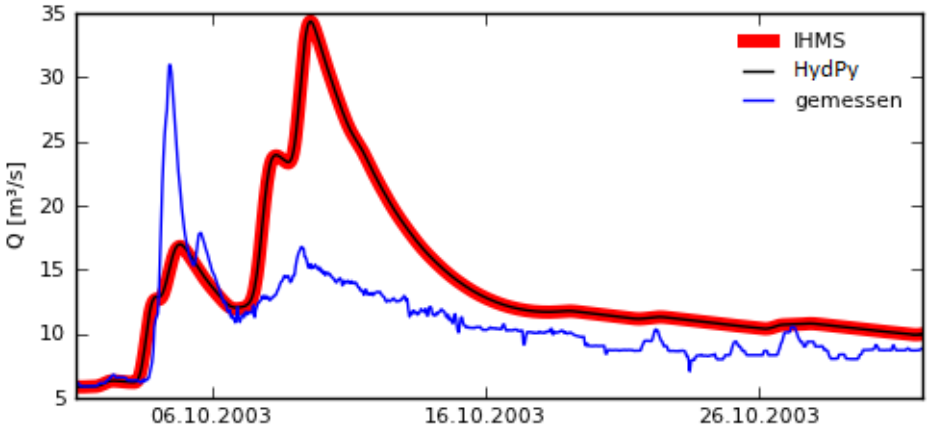


Abbildung 2.4: Vergleich HBV$_{96}$-IHMS mit HBV$_{96}$-HydPy, Lippe 1

Teilgebiet Emscher

Einen noch stärkeren Flachlandcharakter (88 m. ü. NN) als das Kopfgebiet der Lippe und zudem eine extreme anthropogene Prägung weist das Teilgebiet Emscher auf. Die Emscher wird durch 3 „Field"- und 3-„Forest"-Zonen abgebildet. Die Parametrisierung ist wiederum der vom Teilgebiet Neckar 1 vergleichbar. Eine Besonderheit ist, dass auf die Möglichkeit der Abflusserhöhung zurückgegriffen wird (Gleichung 2.45): um Überleitungen aus anderen Flussgebieten zu berücksichtigen, werden konstant 7,5 m³/s zum Teilgebietsausfluss hinzuaddiert. Der gemessene MQ beträgt 15,38 m³/s, die simulierten MQs betragen jeweils 14,42 m³/s. Die Quantile der Residuen sind in Tabelle 2.4 gegeben. Abbildung 2.5 zeigt wiederum eine im Vergleich zu den Messungen sehr gute Übereinstimmung der beiden Modell-Implementierungen.

Tabelle 2.4: Vergleich HBV$_{96}$-IHMS mit HBV$_{96}$-HydPy, Emscher

	Mini-	Quantile							Maxi-
	mum	0.01	0.05	0.1	0.5	0.9	0.95	0.99	mum
ε [m³/s]	-1.02	-0.17	-0.04	-0.01	0.00	0.05	0.08	0.19	1.87
ε [%]	-1.38	-0.49	-0.18	-0.11	-0.02	0.29	0.43	0.81	12.08

Bisher wurde mit dem Gebietsausfluss die Gesamtreaktionen einzelner Kopfgebiete entsprechend HBV$_{96}$-IHMS und HBV$_{96}$-HydPy verglichen. Im Folgenden wird nun die Übereinstimmung der systeminternen Dynamik durch den Vergleich des zeitlichen Verlaufes der Speicherzustände exemplarisch für Teilgebiet Neckar 1 demonstriert. Zonenweise vorliegende Speicher wie *SM* sind flächengewichtet aggregiert dargestellt. Der gewählte Zeitausschnitt ist, vom Beginn der vorhandenen Datenreihen ausgehend, an die Dynamik der jeweiligen Speichergröße angepasst. Weder die Interzeptionsspeicherung (*IC*, Abbildung 2.6), die Speicherung von flüssigem (*WC*, Abbildung 2.7) und gefrorenem (*SP*, Abbildung 2.8) Wasser in der Schneedecke, die Bodenwasserspeicherung (*SM*, Abbildung 2.9) noch die Speicherung des Wassers im oberflächennahen (*UZ*, Abbildung 2.10) und tiefen (*LZ*, Abbildung 2.11) Grundwasser weichen in relevantem Maße voneinander ab.

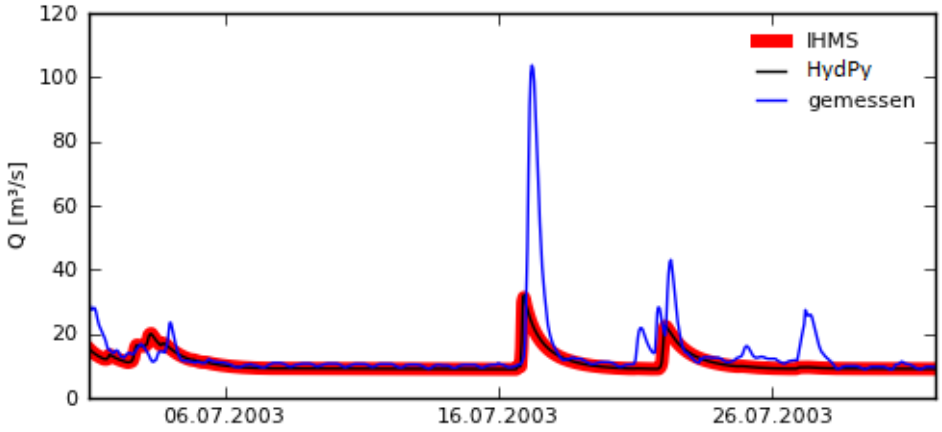


Abbildung 2.5: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Emscher

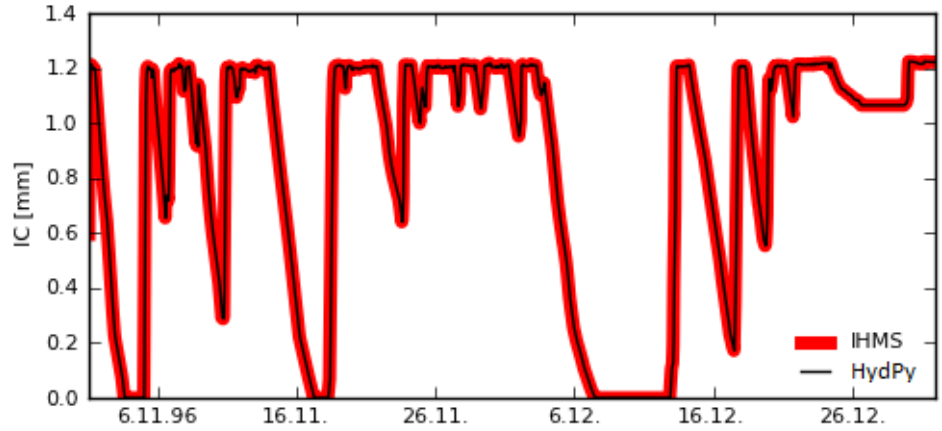


Abbildung 2.6: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Interzeption

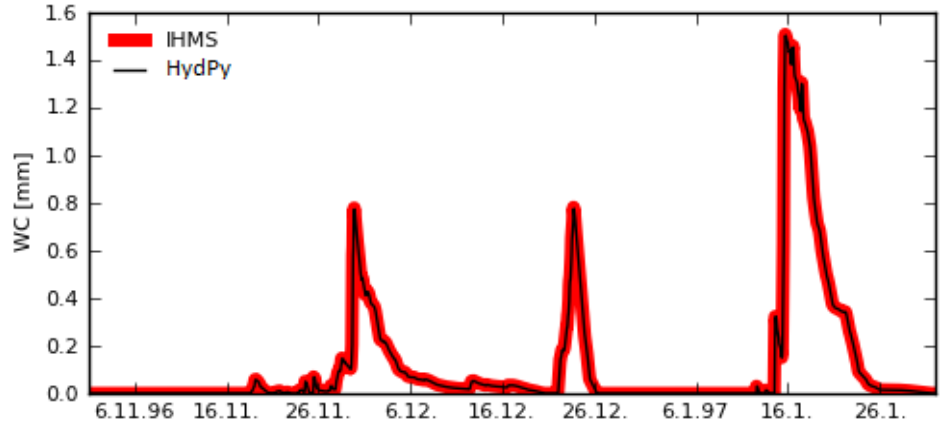


Abbildung 2.7: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, flüssiger Schneedeckenanteil

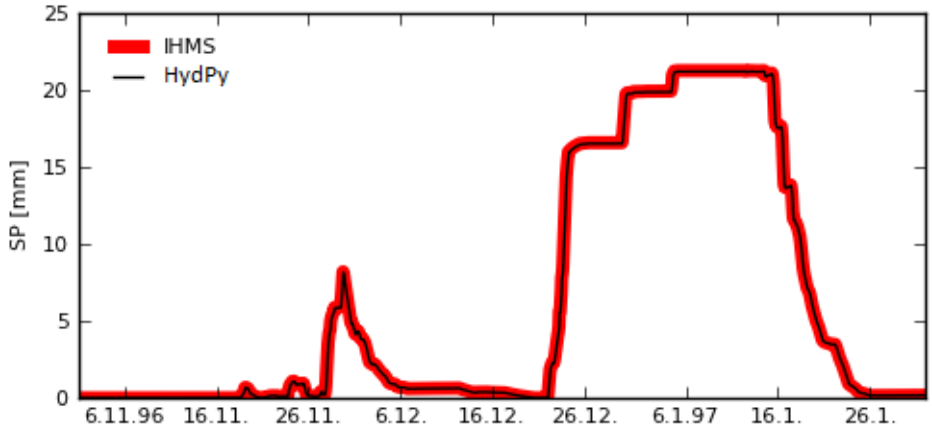


Abbildung 2.8: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, fester Schneedeckenanteil

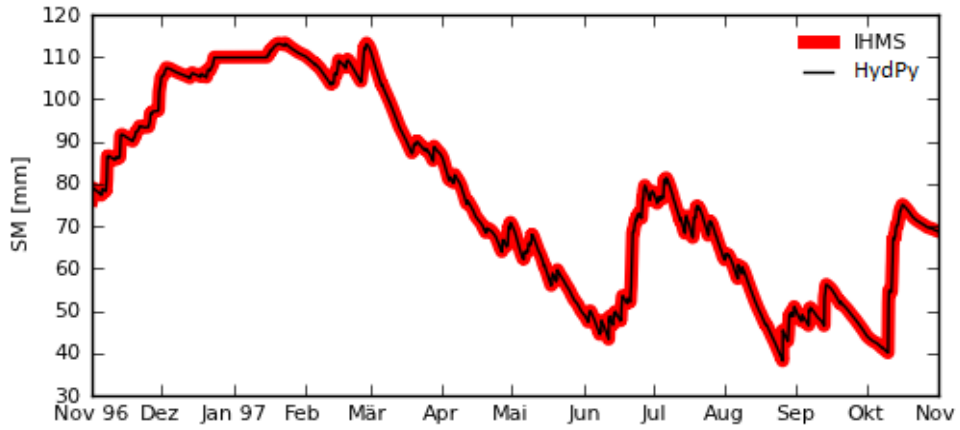


Abbildung 2.9: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Bodenwasser

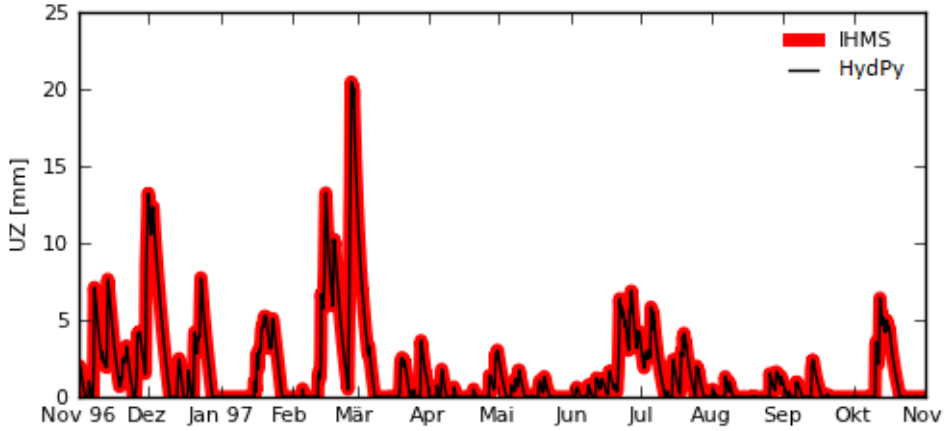


Abbildung 2.10: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, oberflächennahes Grundwasser

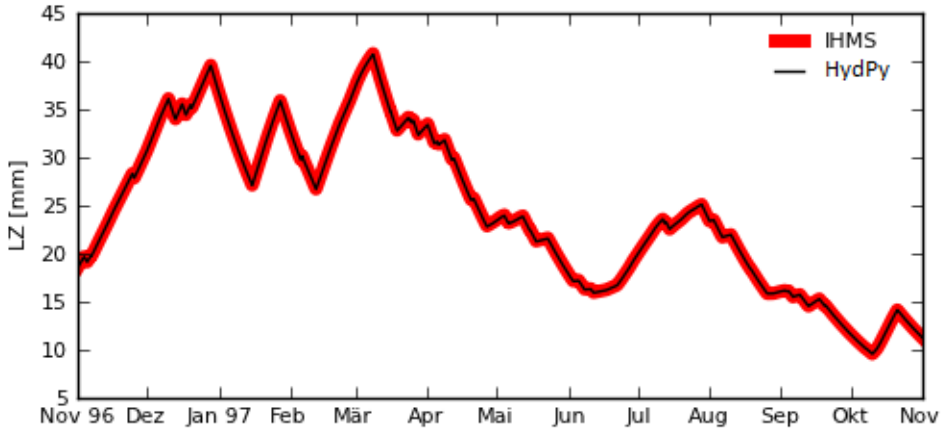


Abbildung 2.11: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, tiefes Grundwasser

Tabelle 2.5 fasst den Grad der Übereinstimmung der mit beiden HBV_{96}-Implementierungen simulierten Abflüsse für alle Teilgebiete zusammen. Als Vergleichskriterium wurde der Koeffizient nach Nash und Sutcliff (Gleichung 2.49) gewählt, da dieser sowohl Volumen- als auch Formfehler erfasst. Die Teilgebietsnamen entsprechen den in der Rhein-BfG-Implementierung verwendeten Abkürzungen. Die Teilgebiete sind, mit der schlechtesten Übereinstimmung beginnend (Rednitz mit NS gleich 0,41894) aufsteigend sortiert. 70 Teilgebiete sind Kopfgebiete, bei denen allein die Programmierung der Zonen-Prozesse relevant ist. Die 65 übrigen Gebiete haben wenigstens ein Oberlieger-Gebiet, womit auch die Programmierung des Wellenablaufes den Grad der Übereinstimmung beeinflusst.

$$NS = 1 - \frac{\left(\vec{Q}_{SMHI} - \vec{Q}_{HWU}\right)^2}{\left(\vec{Q}_{SMHI} - MQ_{SMHI}\right)^2} \qquad 2.49$$

mit:	NS	Nash-Sutcliffe-Koeffizient	[-]
	MQ_{SMHI}	mittlerer simulierter Abfluss nach SMHI	[m³/s]

In der Mehrzahl der Fälle ist der NS-Koeffizient deutlich größer als 0,99, was als ausreichend einzustufen ist. In 11 Gebieten ist die Übereinstimmung dagegen mangelhaft. Die Abweichungen gehen hauptsächlich von Zuflüssen des Main aus (Aisch, Rednitz, Pegnitz, Regnitz). Die einzelnen Abschnitte des Main selbst zeigen umso geringere Abweichungen, je weiter stromab sie sich befinden, d. h. je geringer der relative Einfluss der betroffenen Kopfgebiete ist. Ebenfalls große Abweichungen zeigt das aus Weschnitz und Modau zusammengesetzte Modell-Teilgebiet „Weschmod“. Sowohl Weschnitz und Modau als auch der Main entwässern in das Rhein-Teilgebiet „uprhine4“. „uprhine4“. Diese Gebiete und das unterhalb gelegene „midrhine1“ zeigen aufgrund der großen übrigen Einzugsgebietsfläche aber nur geringfügige Abweichungen.

2.5: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, alle Teilgebiete

EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]
rednitz	0.41894	elzdreis2	0.99936	omos1	0.99988	ruhr4	0.99993	ruhr2	0.99996
weschmod	0.47574	alzette	0.99940	main2	0.99988	ahr	0.99993	schwarzw	0.99996
aisch	0.57539	fils	0.99941	neckar5	0.99988	thuner_s	0.99993	kanal	0.99996
pegnitz	0.64268	ill2	0.99949	omos4	0.99988	blies_1	0.99993	zorn	0.99996
regnitz	0.89898	ill3	0.99951	umos2	0.99989	wied	0.99993	lahn4	0.99996
main3	0.90937	ill1	0.99952	erft_1	0.99989	enz2	0.99993	lahn5	0.99996
main4	0.91937	bruche	0.99953	umos3	0.99989	kl_emme	0.99993	jagst	0.99997
main5	0.95230	neckar2	0.99959	misi	0.99989	omos2	0.99993	rhein3	0.99997
main6	0.96298	nims	0.99967	umos4	0.99989	uprhine3	0.99993	rhein1	0.99997
main7	0.97164	ruwer	0.99976	albpfinz	0.99989	worms	0.99993	uprh2_2	0.99997
main8	0.97517	elsenz	0.99977	obsi	0.99989	nahe3	0.99993	lippe3	0.99997
uprhine4	0.99779	sauer1	0.99978	erft_2	0.99990	thur	0.99994	rhein2	0.99997
midrhine1	0.99795	neckar3	0.99979	erft_3	0.99990	wisper	0.99994	queichsp	0.99997
nied_1	0.99818	sauer2	0.99981	main1	0.99990	our	0.99994	aare1	0.99998
murgren	0.99846	sure	0.99982	wupper1	0.99991	sauwies	0.99994	lippe2	0.99998
fecht	0.99888	seille	0.99983	frsaale	0.99991	agger	0.99994	zurich_s	0.99998
midrhine2	0.99900	unsaar	0.99983	omos3	0.99991	kyll	0.99995	birs	0.99998
saynbach	0.99901	nidda	0.99984	lippe1	0.99991	nahe1	0.99995	bodensee	0.99998
midrhine3	0.99902	rest_1	0.99984	prims_1	0.99992	nette	0.99995	tauber	0.99998
kinzigup	0.99905	lieser	0.99985	wupper2	0.99992	lahn2	0.99995	lim_reus	0.99998
midrhine4	0.99909	murr	0.99985	nahe2	0.99992	dill	0.99995	bieler_s	0.99998
lowrhine1	0.99912	ruhr1	0.99986	unsi	0.99992	enz1	0.99995	selz	0.99998
elzdreis1	0.99915	lahn1	0.99987	neckar1	0.99992	kinzig	0.99995	uprh2_1	0.99998
lowrhine2	0.99919	neckar4	0.99987	uprhine2	0.99992	uprh2_3	0.99995	vierwa_s	0.99998
lowrhine3	0.99922	kocher	0.99987	ruhr3	0.99993	uprhine1	0.99995	rhein4	0.99998
lowrhine4	0.99923	rems	0.99987	obsa	0.99993	emscher	0.99995	aare2	0.99998
orne	0.99931	umos1	0.99987	pruem	0.99993	emme	0.99996	moder	0.99999

Die Ursache der Abweichungen ist in allen fünf Fällen, dass die mit der IHMS-Umsetzung berechnete Verdunstung die zu erwartenden Werte sehr deutlich übersteigt. Sowohl im Vergleich zu anderen, mit der IHMS-Umsetzung simulierten Teilgebieten, als auch im Vergleich mit der HydPy-Umsetzung treten hier Verdunstungswerte auf, die deutlich zu hoch sind. Bislang können weder der Bearbeiter noch die Ansprechpartner der BfG dieses Verhalten der IHMS-Umsetzung mit der Modell-Parametrisierung oder den meteorologischen (Norm-)Eingangsdaten in Einklang bringen.

Im Folgenden ein Zahlenbeispiel für das Teilgebiet Aisch zur Erläuterung. Berechnet wird die potenzielle Verdunstung für den 01.11.1996, 12 Uhr. Das ist der erste Simulationstermin im betrachteten Zeitraum, in dem kein Niederschlag auftritt, was die Berechnung verkürzt. Eine Schneedecke aus vorherigen Zeitschritten ist nicht vorhanden. Das Teilgebiet besteht 10 Zonen, deren Höhenniveaus und Flächengrößen in Tabelle 2.6 gegeben sind.

2.6: Zonenhöhen und -flächen, Teilgebiet Aisch

Zonenindex [-]	1	2	3	4	5	6	7	8	9	10
Höhe ü. NN [m]	300	300	400	400	500	500	600	600	700	700
Teilfläche [km²]	56.12	14.76	232.51	181.3	179.82	142.62	46.17	39.89	0.61	0.6

Die gegebene Temperatur beträgt 11,9 °C, der Temperatur-Höhengradient *TCALT* 0,6 °C/100 m und das Temperatur-Bezugsniveau *zrelT* 335 m. ü. NN. Nach Gleichung 2.1 ergibt sich für die erste Zone eine Temperatur von $11{,}9 - 0{,}6 \cdot (300 - 335) / 100 = 12{,}11°C$.

Die mittlere Temperatur aller Zonen ergibt sich nach Flächengewichtung entsprechend Gleichung 2.2 zu $(56{,}12 \cdot 12{,}11 + ... + 0{,}6 \cdot 9{,}71) / (56{,}12 + ... + 0{,}6) = 11{,}223°C$.

Da kein Niederschlag auftritt, sind hier die Gleichungen 2.3 bis 2.5 irrelevant.

Die Normwerte der Temperatur und der potenziellen Verdunstung sind 4,544 °C und 0,467 mm/d bzw. 0,019 mm/h. Mit einem Temperaturfaktor der Verdunstung von 0,1 ergibt sich nach Gleichung 2.6 ein potenzieller Verdunstungswert von $0{,}02 \cdot (1 + 0{,}1 \cdot (1{,}22 - 4{,}54)) = 0{,}032 mm/h$ bzw. 0,778 mm/d.

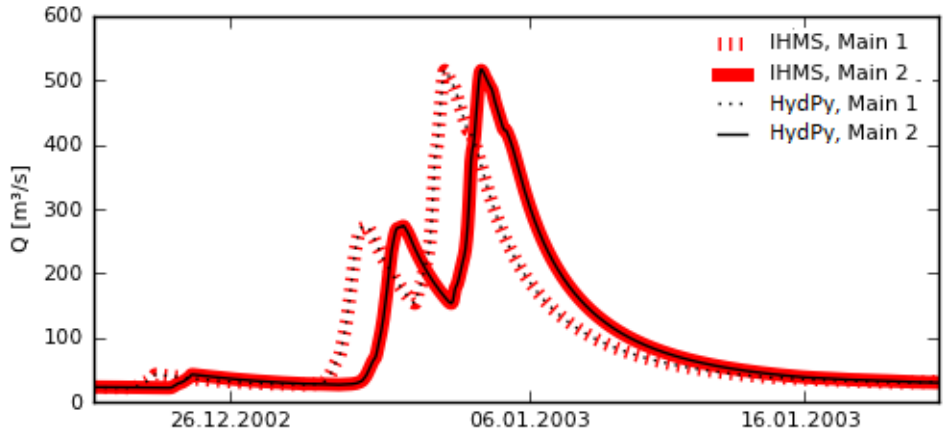
Weitere Korrekturen nach Gleichung 2.7 entfallen, da der allgemeine Verdunstungsfaktor *ECORR* gleich null und der Verdunstungs-Höhengradient *ECALT* gleich eins ist und kein Niederschlag im betreffenden Zeitschritt fällt. Da ERED gleich null ist, kann nach Gleichung 2.29 die tatsächliche Verdunstung maximal der doppelten potenziellen Verdunstung entsprechen und folglich maximal 0,065 mm/h betragen.

Nach der IHMS-Umsetzung wird der berechnete Wert der potenziellen Verdunstung mit 0,058 mm/h angegeben. Die Verdunstung aus den Interzeptionsspeicher *IC* beträgt 0,058 mm/h, die aus dem Bodenspeicher *SM* 0,028 mm/h, was einer Gesamtverdunstung von 0,086 mm/h entspricht.

Um die oben getroffenen Modellannahmen als Ursache für die erhöhten Verdunstungswerte auszuschließen, wird im Folgenden ein Vergleich der Teilgebiete Aisch und Main 1 angestellt. Beide Teilgebiete weisen identische Werte für sämtliche für die Berechnung der potenziellen Verdunstung relevanten Parameter auf. Die meteorologischen Eingangsgrößen zeigen, wie aufgrund der räumlichen Nähe beider Teilgebiete zu erwarten, ähnliche Mittelwerte. So beträgt beispielsweise die mittlere Normtemperatur für Aisch 7,2 °C und für Main 1 6,6 °C. Die potenzielle Jahresverdunstung beträgt im neunjährigen Betrachtungszeitraum laut der IHMS-Implementierung dagegen 664 mm für Main 1 und 1.166 mm für Aisch. Derart hohe Werte der potenziellen Verdunstung waren während des Modellaufbaus sicherlich nicht angestrebt. Fehler in der IHMS-Programmierung oder in der Formatierung der Steuerdateien der Rhein-Konfiguration von HBV_{96} sind daher als mögliche Ursachen anzunehmen.

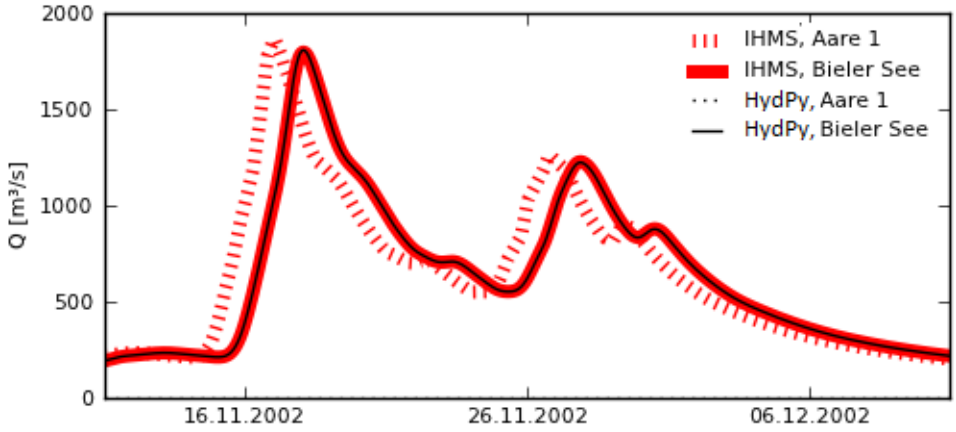
Abschließend wird die Übereinstimmung der Wellenablaufberechnungen der IHMS- und der HydPy-Implementierung anhand dreier Flussstrecken demonstriert. In den folgenden Abbildungen sind für drei Flussabschnitte jeweils die größten simulierten Hochwasser in den hydrologischen Jahren 1997 bis 2005 dargestellt. Für die Oberlieger-Gebiete Main 1, Aare 1 und „UpRhine 2“ (oberer Rhein 2) sind jeweils die Gesamtabflüsse abgebildet. Um nur den Einfluss der Wellenablaufberechnung zu zeigen, sind für die Unterlieger-Gebiete Main 2, Bieler See und „UpRhine 3“ nur die Ausflüsse aus dem jeweiligen Flusslauf dargestellt – die Ausflüsse der Zonen sind damit außen vor.

Im Gewässerabschnitt im Teilgebiet Main 2 findet eine Verzögerung von 32 Stunden (*LAG* gleich 1,35), aber keine Verformung (*DAMP* gleich 0) der Hochwasserwelle Januar 2003 statt. Abbildung 2.12 zeigt keine nennenswerten Abweichungen zwischen der IHMS- und der HydPy-Implementierung.



2.12: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Wellenablauf Main 1 – Main 2

Im Gewässerabschnitt des Teilgebietes Aare 1 kommt es sowohl zu einer zeitlichen Verzögerung von 24 Stunden (*LAG* gleich 1) als auch zu einer geringfügigen Abflachung (*DAMP* gleich 0,5) der Hochwasserwelle im November 2002. Abbildung 2.13 zeigt ebenfalls keinerlei relevante Abweichungen zwischen beiden HBV_{96}-Implementierungen.

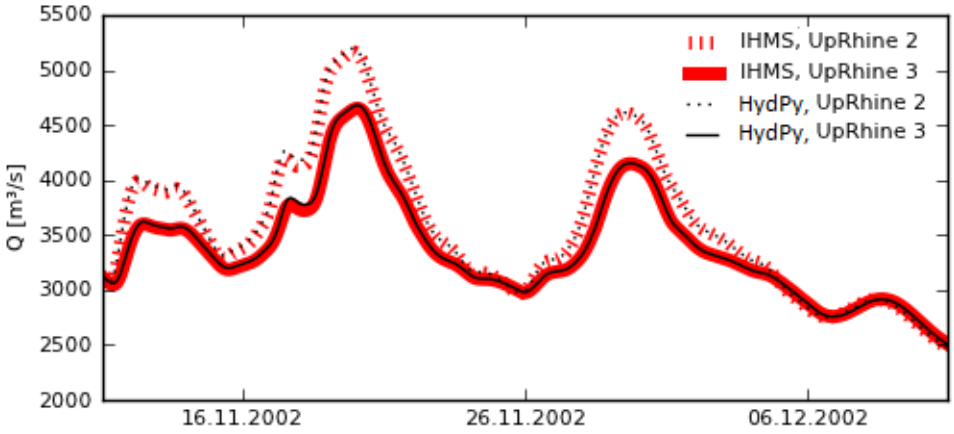


2.13: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Wellenablauf Aare 1 – Bieler See

Im Gewässerabschnitt des Teilgebietes „UpRhine 3“ finden eine geringfügige zeitliche Verzögerung von 4 Stunden (*LAG* gleich 0,167) und eine damit zwangsläufig sehr geringe Abflachung (*DAMP* gleich 0,5) des Hochwassers im November 2002 statt. Zusätzlich kommt es zu einem Volumenverlust. In der IHMS-Implementierung wird oberhalb von 3.000 m³/s ein Teil des Abflusses von Teilgebiet „UpRhine 2“ in das künstliche Teilgebiet „Up3_HW“ abgeführt. „Up3_HW“ ist, da es seine Zuflüsse an kein Teilgebiet weiterreicht, eine Abflusssenke. In der HydPy-Implementierung kann auf derartige künstliche Teilgebiete verzichtet werden. Der Flusslauf-Instanz des Teilgebietes „UpRhine 3“ greift mit Hilfe der Stützstellen-Vektoren *Qtotal* und *Qbranch* nur den gewünschten Abflussanteil ab. Der Zusammenhang zwischen dem Zufluss aus „UpRhine 2“ (*Qtotal*) und dem tatsächlichen Zufluss zu „UpRhine 3“ (*Qbranch*) ist in Tabelle 2.7 gegeben. Zu Beginn des in 2.14 dargestellten Hochwassers überschreiten die mit der HydPy-Implementierung simulierten Abflusswerte die der IHMS-Implementierung leicht. Die Abweichung steht aber nicht mit dem Wellenablauf innerhalb des Teilgebietes „UpRhine 3“ in Verbindung, da Zu- und Abfluss gleichermaßen betroffen sind.

2.7: Braching-Stützstellen für die Teilgebiete „UpRhine 2“ und „UpRhine 3“

Qtotal [m³/s]	0	3.000	4.000	9.000	50.000
Qbranch [m³/s]	0	3.000	3.600	8.100	45.000



2.14: Vergleich HBV_{96}-IHMS mit HBV_{96}-HydPy, Wellenablauf UpRhine 2 – UpRhine 3

Insgesamt zeigt der Vergleich zwischen der IHMS- und der HydPy-Implementierung von HBV_{96}, dass in fast allen Fällen keine praktisch relevanten Differenzen bezüglich der Ausgaben und internen Zustände der Zonen- und Flussabschnitt-Umsetzungen bestehen. Ausgenommen hiervon sind lediglich die Main-Teilgebiete Aisch, Rednitz, Pegnitz und Regnitz sowie das aus Teilen der Weschnitz und der Modau zusammengesetzte Teilgebiet „WeschMod“. In diesen Gebieten sind die mit der IHMS-Implementierung simulierten Werte der potenziellen Verdunstung unplausibel hoch. Die Ursache hierfür ist weder aus den Modelldaten noch der Modellparametrisierung nachzuvollziehen. Der Vollständigkeit halber sei erwähnt, dass sich weitere, wenn auch geringe, systematische Abweichungen zwischen beiden Implementierungen zeigen lassen, beispielsweise im Schneemodul. Diese wurden aber nicht in die obige Darstellung einbezogen, weil sie nicht praxisrelevant sind. Zudem sind Schlüsse bezüglich der Ursachen der Unterschiede schwierig, da die Ausgabemöglichkeiten interner Flüsse und Zustände bei IHMS begrenzt sind. Schlussendlich ist die Übereinstimmung zwischen beider HBV_{96}-Implementierung als ausreichend einzuschätzen, um die Untersuchungen der verschiedenen Teilprojekte mit Hilfe des HydPy-Frameworks und unter Rückgriff auf die Modellparametrisierung und -strukturierung der Rhein-IHMS-Umsetzung der BfG durchführen zu können. Lediglich für die genannten Teilgebiete mit erhöhter Verdunstung ist eine Korrektur bzw. eine Nachkalibrierung erforderlich.

2.2 HBV_{96} (Zustandsraum-Strategie)

Die Lösung des Differenzialgleichungs-Systems der weitgehend originalgetreuen HBV_{96}-Implementierung (Abschnitt 2.1) entspricht der im Abschnitt 1.11 diskutierten ad-hoc-Strategie. Um diese hinsichtlich ihrer Genauigkeit zu bewerten, wurde HBV_{96} zusätzlich in der Zustandsraum-Strategie formuliert. Betroffen sind nur die in den Abschnitten 2.1.2 bis 2.1.5 beschriebenen Modellkomponenten Interzeption, Schnee und Gletscher, Boden sowie Grundwasser. Die Berechnung bzw. Modifizierung der meteorologischen Randbedingungen (Abschnitt 2.1.1), der Wellenablauf (Abschnitt 2.1.7) sowie der Dreiecks-Unit-Hydrograph (Ende von Abschnitt 2.1.5) bleiben unverändert, da hierbei keine Differenzialgleichungen zugrunde liegen.

Die folgenden Beschreibungen der einzelnen Module vertiefen lediglich die Änderungen gegenüber der ad-hoc-Variante von HBV_{96} und setzen somit die Kenntnis von Abschnitt 2.1 voraus. Nur neue Variablennamen werden erläutert.

2.2.1 Interzeption

Die Bedingung für das Auftreten von Throughfall ist die vollständige Füllung des Interzeptionspeichers. Trifft die Bedingung zu, entspricht laut Gleichung 2.8 die Intensität des Throughfalls der des Niederschlages. Gepaart mit der sequentiellen Rechenstrategie entsprechend Gleichung 2.8 bis 2.11 führt dies zur Priorisierung des Throughfalls gegenüber der Interzeptionsverdunstung. In Gleichung 2.50 entspricht die Intensität des Throughfall dagegen der Differenz aus Niederschlag und Interzeptionsverdunstung, was die reale Gleichzeitigkeit aller Wasserflüsse abbildet.

$$TF_z^k = \begin{cases} \max\left(Pc_i^k - EI_z^k;\ 0\right) & falls \quad IC_z^k = ICMAX^k \\ 0 & falls \quad IC_z^k < ICMAX^k \end{cases} \qquad 2.50$$

mit: z Zwischenschritt-Index [-]

Der in Gleichung 2.50 eingeführte Zwischenschritt-Index *z* ersetzt im Folgenden den Zeitschrittindex *i* für diejenigen Größen, die innerhalb eines Zeitschrittes variabel sind. So werden Eingangsgrößen wie der Niederschlag i. d. R. als konstant über ein Mess- bzw. Zeitschrittintervall angenommen. Alle Speichergrößen sowie die speicherabhängigen Flussgrößen werden dagegen – die Formulierung der Modellgleichungen als Differenzialgleichungs-System vorausgesetzt – als zeitvariabel betrachtet.

Der Zwischenschritt-Index *z* steht in Verbindung zum Zeitpunkt des Systemzustandes und der dann auftretenden Flussintensitäten, ist damit aber nicht identisch. Numerische Integrationsalgorithmen wie das 4-stufige Runge-Kutta-Verfahren schätzen die mittlere Flussintensität über einen Zeitabschnitt durch eine gewichtete Mittelung von Flussintensitäten, die aus verschiedenen Systemzuständen für zum Teil identische Zeitpunkte resultieren. Einem Zeitpunkt können somit mehrere Rechen-Zwischenschritte zugeordnet sein.

Interzeptionsverdunstung kann nach Gleichung 2.51 auftreten, wenn der Interzeptionsspeicher gefüllt ist oder wenn Niederschlag fällt. Die Minimum-Funktion gewährleistet bei leerem Interzeptionsspeicher und falls die potenzielle Verdunstung den korrigierten Niederschlag überschreitet, dass die gespeicherte Wassermenge in jedem Rechenschritt eines Zeitintervalls null bleibt, was die numerische Integration erleichtert.

$$EI_z^k = \begin{cases} EPc_i^k & falls \quad IC_z^k > 0 \\ \min\left(Pc_i^k;\ EPc_i^k\right) & falls \quad IC_z^k = 0 \end{cases} \qquad 2.51$$

Die Aktualisierung des Interzeptionsspeichers über eine Rechenschrittweite Δ erfolgt entsprechend Gleichung 2.52. τ verweist auf den Zeitpunkt des Ausgangszustandes, τ + Δ auf den Zeitpunkt des Endzustandes und $\tau \rightarrow \tau + \Delta$ auf das dazwischen liegende Zeitintervall. Die Einklammerung von τ + Δ weist darauf hin, dass der entsprechende Speicherinhalt im Folgenden evtl. korrigiert wird.

$$IC^k_{(\tau+\Delta)} = IC^k_{\tau} + \Delta \cdot \left(Pc^k_i - TF^k_{\tau \rightarrow \tau+\Delta} - EI^k_{\tau \rightarrow \tau+\Delta} \right) \qquad 2.52$$

mit: τ Zwischenschritt-Zeitpunkt [h]

Δ Rechenschrittweite [h]

Die entkoppelte Berechnung von Throughfall und Interzeptionsverdunstung auf der einen sowie der neuen Interzeptionsspeicherung auf der anderen Seite führt zwangsläufig zu Restriktionsverletzungen. Diese werden nach Gleichung 2.53 korrigiert, wobei die Differenz zwischen $IC_{\tau+\Delta}$ und $IC_{(\tau+\Delta)}$ wie in den Abschnitten 1.11.1 und 1.13.2.6 beschrieben zur Justierung der Rechenschrittweite bzw. des Rechenfehlers herangezogen wird.

$$IC^k_{\tau+\Delta} = \min\left(\max\left(IC^k_{(\tau+\Delta)};\ 0 \right);\ ICMAX \right) \qquad 2.53$$

2.2.2 Schnee und Gletscher

Der feste Teil des Throughfalls ${}_{SP}TF$, welcher in den Eisgehalt der Schneedecke eingeht, und der flüssige Teil des Througfalls ${}_{WC}TF$, welcher in den Wassergehalt der Schneedecke eingeht, berechnen sich äquivalent zu den Gleichungen 2.14 und 2.13 nach den Gleichungen 2.54 und 2.55.

$${}_{SP}TF^k_z = TF^k_z \cdot \frac{\left(1 - fracRain^k_i\right) \cdot SFCF}{fracRain^k_i \cdot RFCF + \left(1 - fracRain\right)^k_i \cdot SFCF} \qquad 2.54$$

mit: ${}_{SP}$TF fester Throughfall [mm/h]

$${}_{WC}TF^k_z = TF^k_z \cdot \frac{fracRain^k_i \cdot RFCF}{fracRain^k_i \cdot RFCF + \left(1 - fracRain\right)^k_i \cdot SFCF} \qquad 2.55$$

mit: ${}_{WC}$TF flüssiger Throughfall [mm/h]

Die potenzielle Schmelzrate des Eisanteils der Schneedecke ${}_{pot}MELT$ sowie die potenzielle Gefrierrate des Wasseranteils der Schneedecke ${}_{pot}REFR$ berechnen sich äquivalent zu den Gleichungen 2.15 und 2.16 nach den Gleichungen 2.56 und 2.57.

$$ {}_{pot}MELT_i^k = \max\left(CFMAX \cdot \left(Tc_i^k - TTM\right);\ 0\right) \qquad 2.56 $$

mit: ${}_{pot}$MELT potenzielle Schmelzrate [mm/h]

$$ {}_{pot}REFR_i^k = \max\left(CFR \cdot CFMAX \cdot \left(TTM - Tc_i^k\right);0\right) \qquad 2.57 $$

mit: ${}_{pot}$REFR potenzielle (Wieder-)Gefrierrate [mm/h]

Liegt in der Schneedecke gespeichertes Eis vor, schmilzt dieses entsprechend der potenziellen Schmelzrate (Gleichung 2.58). Liegt keine Schneedecke vor, entspricht die tatsächliche Schmelzrate dem Minimum aus potenzieller Rate und festem Throughfall (vgl. mit der Begründung für Gleichung 2.51).

$$ MELT_z^k = \begin{cases} {}_{pot}MELT_i^k & falls \quad SP_z^k > 0 \\ \min\left({}_{pot}MELT_i^k;\ {}_{SP}TF_z^k\right) & falls \quad SP_z^k = 0 \end{cases} \qquad 2.58 $$

Wie viel in der Schneedecke gespeichertes Wasser tatsächlich gefriert, berechnet sich analog zu Gleichung 2.58 wie folgt (Gleichung 2.59):

$$ REFR_z^k = \begin{cases} {}_{pot}REFR_i^k & falls \quad WC_z^k > 0 \\ \min\left({}_{pot}REFR_i^k;\ {}_{WC}TF_z^k\right) & falls \quad WC_z^k = 0 \end{cases} \qquad 2.59 $$

Die Berechnung des aus der Schneedecke freigesetzten Wassers ist aufgrund der variablen Obergrenze der maximalen Speicherung flüssigen Wassers in der Schneedecke umfangreicher. Die maximale Wasserpeicherung ist zum Eisgehalt der Schneedecke proportional. Hat die tatsächliche Wasserspeicherung den Maximalwert erreicht und ändert sich der Maximalwert aufgrund der zeitlichen Konstanz des Eisgehaltes nicht, wird flüssiger Throughfall unmittelbar an das Bodenmodul weitergereicht. Steigert sich die Maximalspeicherung dagegen aufgrund des niederschlagsbedingten Anstiegs des Eisgehaltes der Schneedecke, verringert sich die Freisetzungsrate proportional zum festen Throughfall. Da die Maximum-Funktion negative Freisetzungen verhindert, tritt ein solcher Fall nur bei Mischniederschlag auf (siehe Gleichung 2.3). Gleichzeitiges Schmelzen und Gefrieren innerhalb der Schneedecke ist aufgrund des übergangslosen Grenzwertes *TTM* in den Gleichungen 2.56 und 2.57 ausgeschlossen. Angenommen es fällt kein Niederschlag und die Maximalspeicherung ist erreicht, führt Schmelze (1) zur unmittelbaren Freisetzung der entsprechenden Wassermenge und (2) zur Freisetzung derjenigen Wassermenge, die aufgrund der Reduzierung des Eisgehaltes zusätzlich nicht mehr von der Schneedecke gehalten wird. Umgekehrt wird bei Maximalspeicherung die Weitergabe von flüssigem Throughfall an das Bodenmodul durch Gefrieren von Wasser entsprechend (1) der Gefrierrate sowie (2) der Steigerung der Maximalspeicherung reduziert.

$$IN_t^k = \begin{cases} \max\left(\begin{matrix} {}_{WC}TF_z^k - WHC \cdot {}_{SP}TF_z^k + \ldots \\ \ldots\left(1+WHC\right)\cdot\left(MELT_z^k - REFR_z^k\right) \end{matrix};\ 0 \right) & falls \quad WC_z^k = \dfrac{SP_z^k}{\left(WHC\right)^{-1}} \\ 0 & falls \quad WC_z^k < \dfrac{SP_z^k}{\left(WHC\right)^{-1}} \end{cases} \qquad 2.60$$

Die Aktualisierung der Speicherzustände und die Korrektur von Restriktionsverletzungen erfolgt entsprechend Gleichung 2.61 bis 2.64. Es ist zu beachten, dass zur Korrektur des Wassergehaltes in Gleichung 2.63 der bereits korrigierte Eisgehalt heranzuziehen ist (oder alternativ die Reihenfolge von Minimum und Maximum-Funktion umzukehren wäre).

$$WC_{(\tau+\Delta)}^k = WC_\tau^k + \Delta \cdot \left({}_{WC}TF_{\tau\to\tau+\Delta}^k + MELT_{\tau\to\tau+\Delta}^k - REFR_{\tau\to\tau+\Delta}^k - IN_{\tau\to\tau+\Delta}^k \right) \qquad 2.61$$

$$SP_{(\tau+\Delta)}^k = SP_\tau^k + \Delta \cdot \left({}_{SP}TF_{\tau\to\tau+\Delta}^k + REFR_{\tau\to\tau+\Delta}^k - MELT_{\tau\to\tau+\Delta}^k \right) \qquad 2.62$$

$$WC_{\tau+\Delta}^k = \min\left(\max\left(WC_{(\tau+\Delta)}^k;\ 0 \right);\ WHC \cdot SP_{\tau+\Delta}^k \right) \qquad 2.63$$

$$SP_{\tau+\Delta}^k = \max\left(SP_{(\tau+\Delta)}^k;\ 0 \right) \qquad 2.64$$

2.2.3 Boden

Gleichung 2.65 berechnet den (potenziellen) Abfluss. Sie ist eine Vereinfachung von Gleichung 2.24 die sich daraus ergibt, dass entsprechend der Zustandsraum-Strategie nur die aktuelle Bodenfeuchte für den Abflussbeiwert ausschlaggebend sein kann.

$$R_z^k = IN_z^k \cdot \left(\frac{SM_z^k}{FC} \right)^{BETA} \qquad 2.65$$

Gleichung 2.66 definiert den potenziellen kapillaren Wiederaufstieg bei gegebener Bodenfeuchte entsprechend Gleichung 2.26.

$${}_{pot}CF_z^k = CFLUX \cdot \left(1 - \frac{SM_z^k}{FC} \right) \qquad 2.66$$

Der tatsächliche kapillare Wiederaufstieg hängt zusätzlich vom Füllstand des oberflächennahen Grundwassers ab (Gleichung 2.67). Die Begründung der Minimum-Funktion entspricht derjenigen zur Minimum-Funktion in Gleichung 2.51.

$$CF_z^k = \begin{cases} {}_{pot}CF_z^k & falls \quad UZ_z > 0 \\ \min\left({}_{pot}CF_z^k;\ R_i^k \right) & falls \quad UZ_z = 0 \end{cases} \qquad 2.67$$

Verdunstung aus dem Bodenkörper kann nur auftreten, wenn die Bodenfeuchte größer null ist und keine Schneedecke vorliegt (Gleichung 2.68). Die Minimum-Funktion dient analog zu Gleichung 2.28 dazu, dass die tatsächliche Verdunstung die potenzielle Verdunstung nicht überschreitet.

$$EA_t^k = \begin{cases} \min\left(EPc_i^k \cdot \frac{SM_z^k}{LP \cdot FC};\ EPc_i^k \right) & falls \quad SP_z^k = 0 \\ 0 & falls \quad SP_z^k > 0 \end{cases} \qquad 2.68$$

Gleichung 2.69 dient zur Reduzierung der Bodenverdunstung aufgrund der ggf. priorisierten Interzeptionsverdunstung und ist bis auf den Zwischenschritt-Index mit Gleichung 2.29 identisch.

$$EA_z^k = EA_t^k - \max\left(ERED \cdot \left(EA_t^k + EI_i^k - EPc_i^k \right);\ 0 \right) \qquad 2.69$$

Gleichung 2.70 aktualisiert die Bodenfeuchte für einen Rechenschritt woraufhin Gleichung 2.71 diese ggf. korrigiert.

$$SM_{(\tau+\Delta)}^k = SM_\tau^k + \Delta \cdot \left(IN_{\tau \to \tau+\Delta}^k + CF_{\tau \to \tau+\Delta}^k - R_{\tau \to \tau+\Delta}^k - EA_{\tau \to \tau+\Delta}^k \right) \qquad 2.70$$

$$SM_{\tau+\Delta}^k = \min\left(\max\left(SM_{(\tau+\Delta)}^k;\ 0 \right);\ \mathrm{FC} \right) \qquad 2.71$$

2.2.4 Abflusskonzentration bzw. Grundwasser

Der Nettozufluss zum oberen Grundwasserspeicher *INUZ* berechnet sich analog zu Gleichung 2.32 wie folgt (Gleichung 2.72):

$$INUZ_z = \sum_{k \in Z_L} \left(\frac{A^k}{\sum_{k \in Z_L} A^k} \cdot \left(R_z^k - CF_z^k \right) \right) \qquad 2.72$$

mit: INUZ Nettozufluss zum oberen Grundwasser [mm/h]

Gleichung 2.73 zur Berechnung der contributing area ist bis auf die Indices mit Gleichung 2.33 identisch.

$$contriArea_z = \begin{cases} \sum_{k \in Z_S} \left(\frac{A^k}{\sum_{k \in Z_S} A^k} \cdot \left(\frac{SM_z^k}{FC} \right)^{BETA} \right) & falls \quad (RESPAREA = 1) \cap (Z_S \neq \varnothing) \\ 1 & falls \quad (RESPAREA = 0) \cup (Z_S = \varnothing) \end{cases} \quad 2.73$$

Die Berechnung der Perkolation vom oberen zum unteren Grundwasser nach Gleichung 2.74 entspricht dem wiederholt verwendeten Schema für Flüsse, die vom Speicherinhalt nur durch das Vorliegen eines unteren Grenzwertes abhängen.

$$PERC_z = \begin{cases} PERCMAX \cdot contriArea_z & falls \quad UZ_z > 0 \\ \min\left(PERCMAX \cdot contriArea_z;\ INUZ_z\right) & falls \quad UZ_z = 0 \end{cases} \quad 2.74$$

Gleichung 2.75 zur Berechnung des Direktabflusses entspricht Gleichung 2.36.

$$Q0_z = K \cdot \left(\frac{UZ_z}{contriArea_z} \right)^{ALPHA+1} \quad 2.75$$

Die Änderung und Korrektur des oberen Grundwasserspeichers ergeben sich nach Gleichung 2.76 und 2.77.

$$UZ_{(\tau+\Delta)} = UZ_\tau + \Delta \cdot \left(INUZ_{\tau \to \tau+\Delta} - PERC_{\tau \to \tau+\Delta} - Q0_{\tau \to \tau+\Delta} \right) \quad 2.76$$

$$UZ_{\tau+\Delta}^k = \max\left(UZ_{(\tau+\Delta)}^k;\ 0 \right) \quad 2.77$$

Gleichung 2.78 berechnet den Basisabfluss und ist bis auf die Indizes mit Gleichung 2.41 identisch.

$$Q1_z = \max\left(K4 \cdot LZ_z;\ 0 \right) \quad 2.78$$

Die Aktualisierung des tiefen Grundwassers entsprechend den verschiedenen Flächenrelationen nach Gleichung 2.79 erfolgt analog zu den Gleichungen 2.39, 2.40 und 2.42. Eine Korrektur ist nicht notwendig, da für das tiefe Grundwasser weder ein unterer noch ein oberer Grenzwert vorgesehen ist.

$$LZ_{\tau+\Delta} = \begin{cases} LZ_{\tau} + \Delta \cdot \left(\dfrac{\sum\limits_{k \in Z_L} A^k}{\sum\limits_{k \in Z} A^k} \cdot PERC_{\tau \to \tau+\Delta} - Q1_{\tau \to \tau+\Delta} + \ldots \right. \\ \left. \ldots \sum\limits_{k \in Z_l} \left(\dfrac{A^k}{\sum\limits_{k \in Z} A^k} \cdot R^k_{\tau \to \tau+\Delta} \right) + \sum\limits_{k \in Z_l | Tc_i^k > TTICE} \left(\dfrac{A^k}{\sum\limits_{k \in Z} A^k} \cdot \left(Pc_i^k - EPc_i^k \right) \right) \right) \end{cases} \quad 2.79$$

2.2.5 Vergleich mit HBV_{96} in der ad-hoc-Umsetzung

Wie im Abschnitt 2.1.8 werden sämtliche Vergleichsrechnungen entsprechend der Parametrisierung und den Anfangsbedingungen des BfG-Vorhersagemodells für den Rhein durchgeführt. Einzige Ausnahme ist die Seille. In der Parametrisierung der HBV_{96}-Implementierung der BfG ist *ETF* für sämtliche Zonen des Teilgebietes Seille der Wert null zugewiesen. Diese Parameterbelegung ist für die in Abschnitt 2.3 beschrieben HBV-Variante aufgrund der Verwendung von *smooth3* in der Verdunstungsberechnung ausgeschlossen (Gleichung 2.86). Um die Konsistenz der folgenden Simulationsrechnungen zu gewährleisten, wurde *ETF* ersatzweise mit dem Wert 0,1 belegt. Der Simulationszeitraum beginnt am 01.11.1996 und endet am 31.10.2002.

Die Simulationsergebnisse von HBV_{96} in der ad hoc- sowie in der Zustandsraum-Formulierung stimmen weitgehend überein. Das veranschaulicht Tabelle 2.8 anhand von aufsteigend sortierten Nash-Sutcliffe-Koeffizienten, die für sämtliche Teilgebiete beinahe eins betragen. Bei reiner Betrachtung der Systemreaktion sind die Unterschiede beider HBV_{96}-Varianten marginal.

Tabelle 2.8: ad hoc- vs. Zustandsraum-Implementierung von HBV96, alle Teilgebiete

EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]	EZG [-]	NS [-]
nied_1	0.99805	elsenz	0.99966	omos3	0.99978	main6	0.99984	lahn1	0.99990
murgren	0.99823	obsa	0.99967	neckar1	0.99978	jagst	0.99984	nahe3	0.99991
selz	0.99857	rhein2	0.99967	nette	0.99979	uprhine4	0.99985	thuner_s	0.99991
orne	0.99883	nims	0.99967	omos2	0.99979	midrhi-	0.99985	ruhr4	0.99991
fils	0.99897	zorn	0.99967	aisch	0.99979	main8	0.99985	ruhr3	0.99991
kinzigup	0.99898	erft_1	0.99971	regnitz	0.99981	main7	0.99985	kanal	0.99991
neckar2	0.99909	umos1	0.99971	tauber	0.99981	prims_1	0.99985	lieser	0.99991
erft_3	0.99912	rems	0.99971	lippe3	0.99981	rhein3	0.99985	our	0.99992
erft_2	0.99924	ruwer	0.99971	lippe2	0.99981	wupper2	0.99985	nahe2	0.99992
elzdreis1	0.99925	sauer1	0.99971	main3	0.99981	uprhine3	0.99985	nahe1	0.99992
elzdreis2	0.99929	omos4	0.99972	main4	0.99981	worms	0.99985	kyll	0.99992
alzette	0.99934	weschmod	0.99972	sure	0.99982	unsi	0.99986	dill	0.99993
neckar3	0.99945	murr	0.99972	midrhi-	0.99982	ruhr1	0.99987	uprh2_2	0.99993
bruche	0.99949	lippe1	0.99972	emscher	0.99982	kinzig	0.99987	ruhr2	0.99994
fecht	0.99954	moder	0.99974	saynbach	0.99982	queichsp	0.99987	uprh2_1	0.99995
albpfinz	0.99954	umos2	0.99974	blies_1	0.99982	wupper1	0.99987	rhein4	0.99995
rhein1	0.99955	pegnitz	0.99974	midrhi-	0.99982	wisper	0.99988	aare1	0.99996
ill3	0.99959	umos3	0.99974	midrhi-	0.99983	frsaale	0.99988	bieler_s	0.99996
unsaar	0.99962	umos4	0.99974	lowrhine1	0.99983	main2	0.99989	schwarzw	0.99996
ill2	0.99962	kocher	0.99975	misi	0.99983	uprhine1	0.99989	emme	0.99997
ahr	0.99962	sauer2	0.99975	lowrhine2	0.99983	lahn5	0.99989	kl_emme	0.99997
rednitz	0.99962	wied	0.99976	enz1	0.99984	uprh2_3	0.99989	thur	0.99997
ill1	0.99962	sauwies	0.99976	main5	0.99984	lahn4	0.99989	birs	0.99997
seille	0.99963	omos1	0.99977	lowrhine4	0.99984	pruem	0.99990	aare2	0.99998
neckar4	0.99963	enz2	0.99977	lowrhine3	0.99984	lahn2	0.99990	zurich_s	0.99998
rest_1	0.99963	bodensee	0.99978	uprhine2	0.99984	agger	0.99990	lim_reus	0.99998
neckar5	0.99966	nidda	0.99978	obsi	0.99984	main1	0.99990	vierwa_s	0.99999

Tabelle 2.9: ad hoc- vs. Zustandsraum-Implementierung von HBV96, Neckar 1

Flussgröße	NS-Koeffizient	Abweichung
[-]	[-]	[%]
Pc	1.00000	0.00
TF	0.99925	-0.73
EI	0.98182	1.59
IN	0.99924	-0.73
R	0.99953	-1.14
EA	0.99973	-0.45
PERC	0.99546	-0.39
Q0	0.99978	-1.24
Q1	0.99983	-0.38
Q	0.99978	-0.73

Ein Vergleich des modellinternen Systemverhaltens beider HBV_{96}-Varianten ist in Tabelle 2.9 exemplarisch für das Teilgebiet Neckar 1 dargestellt. Die Angaben zu den ersten sechs Flussgrößen (*Pc* bis *EA*) sind jeweils auf die erste Zone des Teilgebietes bezogen. Die übrigen Flussgrößen (*PERC* bis *Q*) sind gebietsweit konstant. Der in die erste Zone eingehende korrigierte Niederschlag *Pc* ist für beide Modellvarianten identisch, da dieser innerhalb eines Zeitschrittes keine Zeitabhängigkeit aufweist. Entsprechend beträgt der Nash-Sutcliffe-Koeffizient eins und die Abweichung 0 %. Der Throughfall *TF* und insbesondere die Interzeptionsverdunstung *EI* zeigen dagegen nennenswerte relative Abweichungen. In der ad hoc-Implementierung wird *TF* gegenüber *EI* priorisiert. Infolge der simultanen Berechnung beider Flussgrößen nach der Zustandsraum-Strategie ist hierbei TF kleiner und *EI* größer. Im Nash-Sutcliffe-Koeffizienten zeigt sich das aber nur für *EI* deutlich. Die Nash-Sutcliffe-Koeffizienten der auf *TF* folgenden Wasserflüsse (*IN*, *R* usw.) näheren sich sukzessive dem Wert Eins an. Weitere numerische Ungenauigkeiten in der ad hoc-Implementierung sind somit nicht feststellbar. Das kann damit erklärt werden, dass in der ad hoc-Implementierung (1) das Schneemodul auf Zeitschrittbasis exakt gelöst wird (Freisetzung aus der Schneedecke *IN*), (2) die Flüsse des oberflächennahe Grundwassers (*PERC* und *Q0*) mit einer verringerten Rechenschrittweite bestimmt werden und (3) das Bodenmodul (*R* und *EA*) und das tiefe Grundwasser (*Q* und ggf. *EA*) träge auf Speicherinhaltsänderungen reagieren. Im Ergebnis zeigt sich in der ad hoc-Implementierung lediglich ein Bilanzfehler von weniger als einem Prozent, was etwa dem Mittel über alle Gebiete entspricht.

Der Vollständigkeit halber sei angemerkt, dass der Bilanzfehler der ad hoc-Implementierung auch ohne numerische Integration auf Gesamtzeitschrittbasis zu korrigieren ist. Hierfür ist lediglich die Berechnung der Interzeptionsverdunstung derjenigen des Throughfalls vorzuziehen (Gleichungen 2.80 bis 2.83):

$$EI_i^k = \min\left(EPc_i^k; Pc_i^k + \frac{IC_t^k}{dt}\right) \qquad 2.80$$

$$IC_t^k = IC_i^k + dt \cdot \left(Pc_i^k - EI_i^k\right) \qquad 2.81$$

$$TF_i^k = \max\left(\frac{IC_t^k - ICMAX^k}{dt}; 0\right) \qquad 2.82$$

$$IC_j^k = IC_t^k - dt \cdot TF_i^k \qquad 2.83$$

Die ad hoc-Reihenfolge Interzeptionsverdunstung-Throughfall hat aber (ebenso wie die Zustandsraum-Implementierung der Interzeption) zur Folge, dass die Belegung von *ICMAX* mit dem Wert null nicht gleichbedeutend mit der Auslassung sämtlicher Interzeptionsprozesse ist. In Zeiträumen mit Niederschlag tritt weiterhin Interzeptionsverdunstung auf.

2.3 HBV_{exp}

HBV_{exp} basiert auf der in Abschnitt 2.2 beschriebenen Umsetzung von HBV_{96}. Im Unterschied zur Originalkonzeption sind sämtliche Unstetigkeiten im Differenzialgleichungs-System entsprechend der in Abschnitt 1.13.3 dargelegten Methodik beseitigt. HBV_{exp} lässt sich als Verallgemeinerung von HBV_{96} auffassen. Sind die Glättungsgrade der in HBV_{exp} implementierten Glättungsfunktionen gering angesetzt, zeigen beide Modellvarianten ein weitgehend übereinstimmendes Systemverhalten. Hiervon teilweise ausgenommen ist die Abflusskonzentration. Die verwendete Glättungs-Methodik ist auf den Dreiecks-Unit-Hydrograph nicht anwendbar, weshalb anstelle dessen eine Speicherkaskade mit ähnlicher Systemreaktion zum Einsatz kommt. Ein weiterer Grund für den Austausch ist, dass Faltungsoperationen ein Gedächtnis in das Modell einbringen, d. h. Systemreaktionen hängen nicht nur von aktuellen, sondern auch von vorangegangenen Begebenheiten direkt ab. Ein solche übergreifende Zeitabhängigkeit erschwert Modellanwendungen die beispielsweise auf Zustandsnachführungen beruhen.

Im Folgenden werden die Änderungen gegenüber der HBV_{96}-Umsetzung entsprechend der Zustandsraum-Strategie erläutert. Die Kenntnis der Abschnitte 2.1 und 2.2 wird vorausgesetzt. Die Änderungen betreffen hauptsächlich die Berechnungen der meteorologischen Randbedingungen und Flussgrößen. Von der neu eingeführten Speicherkaskade abgesehen, bestehen gegenüber Abschnitt 2.2 keine Unterschiede in den Speicheraktualisierungen. Sämtliche nachgeschalteten Korrekturen entfallen, da die Speichergrößen im Prinzip beliebige Werte annehmen können bzw. nicht scharf durch Restriktionen eingeschränkt sind.

2.3.1 Meteorologische Randbedingungen

In der Aufbereitung der meteorologischen Randbedingungen sind lediglich die Unstetigkeiten in der Berechnung des flüssigen Niederschlaganteils sowie der potenziellen Verdunstung zu eliminieren.

Die Unstetigkeiten in Gleichung 2.3 zur Berechnung des flüssigen Niederschlaganteils drücken sich in der Verschachtelung von Minimum- und Maximum-Funktion aus. Im Ergebnis liegen zwei Grenzwerte vor, zwischen denen eine lineare Interpolation erfolgt. Das entspricht dem unteren Fall in Abbildung 1.8, für den die Glättungsfunktion *smooth3* (Gleichung 1.6 ff.) vorgesehen ist. Da die für *smooth3* notwendige Normierung auf die Grenzwerte null und eins in Gleichung 2.3 bereits enthalten ist, kann der entsprechende Term unmittelbar als Eingangsgröße für *smooth3* übernommen werden (Gleichung 2.84).

$$fracRain_i^k = f_{smooth3}\left(\frac{Tc_i^k - \left(TT - TTINT/2\right)}{TTINT};\ s_3\right) \qquad 2.84$$

mit:	$f_{smooth3}$	Glättungsfunktion smooth3	[-]
	s_3	Spezifischer Glättungsparameter	[-]

Sämtliche Glättungsparameter werden anhand der in Abschnitt 1.13.3.2 beschriebenen Methodik ermittelt. Die Bestimmung der Parameterwerte von s_1 und s_2 wird in den folgenden Einzelfällen nicht explizit erwähnt. s_1 und s_2 ergeben sich unmittelbar aus den Gleichungen 1.11 und 1.12 ff. Zudem sind s_1 und s_2 lediglich vom Metaparameter s_T (für Unstetigkeiten bzgl. der Temperatur) oder s_W (für Unstetigkeiten bzgl. des Wassergehaltes) abhängig, also in jedem der folgenden Anwendungsfälle identisch. Das ist beim Glättungsparameter s_3 nicht der der Fall. Aufgrund der für die Anwendung von *smooth3* notwendigen Normierung der Grenzwerte auf null und eins, ist der Parameter s_3 hinsichtlich der ursprünglichen Intervallbreite der Grenzwerte anzupassen.

Zur Ermittlung des für Gleichung 2.84 spezifischen Glättungsparameters s_3 ist zunächst der allgemeine Metaparameter für temperaturbezogene Unstetigkeiten s_T zu normieren. Hierfür wird er durch das Temperaturintervall mit Mischniederschlag *TTINT* dividiert. s_3 ergibt sich dann durch Umsetzung der Gleichungen 1.12 bis 1.14 mit dem Newton-Rapshon-Verfahren auf den normierten Metaparameter. Diese Vorgehensweise wird im Folgenden als Funktion *p* bezeichnet (Gleichung 2.85).

$$s_3 = p\left(\frac{s_T}{TTINT}\right) \qquad 2.85$$

mit:	p	(iterative) Lösung von Gleichung 1.12	[-]
	s_T	Temperatur-Metaparameter	[°C]

Die Glättung von Gleichung 2.6 zur Berechnung der potenziellen Verdunstung erfolgt ebenfalls mit *smooth3*. Der Term $1 + ETF \cdot \left(Tmean_i - Tn_i\right)$ stellt bereits eine Normierung auf das Intervall null bis zwei dar. Entsprechend ist in Gleichung 2.86 zur Normierung auf die Grenzwerte null und eins lediglich die Division durch zwei ergänzt. Diese wird nach Anwendung von *smooth3* durch die Multiplikation mit dem Faktor zwei rückgängig gemacht.

$$EP_i = EPn_i \cdot 2 \cdot f_{smooth3}\left(\frac{1 + ETF \cdot \left(Tmean_i - Tn_i\right)}{2};\ s_3\right) \qquad 2.86$$

Der spezifische Glättungsparameter s_3 ergibt sich aus dem mittels der ursprünglichen Intervallbreite $2/ETF$ normierten Temperatur-Metaparameter (Gleichung 2.87):

$$s_3 = p\left(\frac{s_T}{2/ETF}\right) \qquad 2.87$$

2.3.2 Interzeption

Die Abbildung der Interzeptionsprozesse in HBV_{96} beruht auf zwei Grenzwerten: Die Interzeptionsspeicherung nimmt minimal den Wert Null und maximal den Wert *ICMAX* an. Diese Grenzwerte steuern das Auftreten von Throughfall und Interzeptionsverdunstung. Eine Abhängigkeit beider Flussgrößen von der Speicherfüllung im Zwischenbereich der Grenzwerte besteht nicht. Das entspricht dem oben in Abbildung 1.8 dargestellten Fall, für dessen Glättung *smooth1* geeignet ist.

Gleichung 2.50 zur Berechnung des Throughfalls wird ersetzt durch Gleichung 2.88. Das hat den Effekt, dass *ICMAX* nicht mehr der größtmögliche Wert der Interzeptionsspeicherung ist sondern derjenige Wert, bei dem Niederschlag die Vegetationsschicht zu 50 % passiert und zu 50 % von dieser zurückgehalten wird. Entsprechend kann die auf der Vegetationsschicht gespeicherte Wassermenge *ICMAX* je nach Wert von s_3 – der sich entsprechend Gleichung 1.11 aus dem Metaparameter für wassermengenabhängige Unstetigkeiten s_W ergibt – mehr oder weniger weit überschreiten.

$$TF_z^k = Pc_i^k \cdot f_{smooth1}\left(IC_z^k - ICMAX;\ s_1\right) \qquad 2.88$$

Zur Glättung der Interzeptionsverdunstung wird Gleichung 2.51 durch Gleichung 2.89 ersetzt. Äquivalent zum Throughfall wird durch die Neuformulierung die zuvor scharfe Grenze von null zu demjenigen Wert, bei dem die Interzeptionsverdunstung 50 % der potenziellen Verdunstung beträgt. Damit einhergehende negative Werte der Interzeptionsspeicherung sind zwar kontraintuitiv. Inhaltlich betrachtet ist die übergangslose Verringerung der Interzeptionsverdunstung bei zunehmender Austrocknung der vielen verschieden zugänglichen Interzeptionsreservoirs eines realen Bestandes dagegen plausibler als der plötzliche Abbruch einer zuvor hohen Verdunstung. Technisch betrachtet steuern nicht die absoluten Werte der unteren und oberen Grenze des Interzeptionsspeichers das Systemverhalten, sondern lediglich deren Differenz.

$$EI_z^k = EPc_i^k \cdot f_{smooth1}\left(IC_z^k;\ s_1\right) \qquad 2.89$$

2.3.3 Schnee und Gletscher

In den Gleichungen zu den Schneeprozessen sind sowohl temperaturbezogene Unstetigkeiten (potenzielles Schmelzen und Gefrieren) als auch speicherbezogene Unstetigkeiten (tatsächliches Schmelzen und Gefrieren sowie die Freisetzung von Wasser aus der Schneedecke) zu eliminieren.

Die Gleichungen 2.56 und 2.57 zur Berechnung der potenziellen Schmelz- und Gefrierraten des Eis- und Wasseranteils der Schneedecke werden durch die Gleichungen 2.90 und 2.91 ersetzt.

Die Unstetigkeit des Originalzusammenhangs entspricht jeweils dem mittleren Fall in Abbildung 1.8, entsprechend kommt die Glättungsfunktion *smooth2* zum Einsatz.

$$ {}_{pot}MELT_i^k = CFMAX \cdot f_{smooth2}\left(Tc_i^k - TTM;\ s_2\right) \qquad 2.90 $$

$$ {}_{pot}REFR_i^k = CFR \cdot CFMAX \cdot f_{smooth2}\left(TTM - Tc_i^k;\ s_2\right) \qquad 2.91 $$

Die Existenz einer Schneedecke bzw. ein Wert von *SP* größer null ist in der HBV_{96}-Konzeption Voraussetzung für das Auftreten von Schneeschmelze und unterdrückt die Gletscherschmelze sowie die Verdunstung aus dem Bodenkörper. Alle resultierenden Unstetigkeiten sind strukturell äquivalent und können mit Hilfe der in Gleichung 2.92 definierten Hilfsgröße ${}^*SP$ eliminiert werden. ${}^*SP$ ist als Wirkungsgrad aufzufassen, welcher z. B. die Bodenverdunstung aufgrund des Eisgehaltes der Schneedecke teilweise reduziert.

$$ {}^*SP_z^k = f_{smooth1}\left(SP_z^k;\ s_1\right) \qquad 2.92 $$

mit: *SP Wirkungsgrad des Schnee-Eisanteils [mm]

Die tatsächliche Schneeschmelze berechnet sich als Produkt der geglätteten potenziellen Schmelzrate mit dem Wirkungsgrad des Eisanteils der Schneedecke (Gleichung 2.93):

$$ MELT_z^k = {}_{pot}MELT_i^k \cdot {}^*SP_z^k \qquad 2.93 $$

Analog zur Schneeschmelze berechnet sich die tatsächliche Gefrierrate als Produkt der geglätteten potenziellen Gefrierrate mit dem Wirkungsgrad des Wasseranteils der Schneedecke (Gleichung 2.94):

$$ REFR_z^k = {}_{pot}REFR_i^k \cdot f_{smooth1}\left(WC_z^k;\ s_1\right) \qquad 2.94 $$

Wie bereits in der Zustandsraum-Formulierung von HBV_{96} (Gleichung 2.60) erweist sich die Berechnung der Wasserfreisetzung aus der Schneedecke als etwas komplizierter (Gleichung 2.95). Die Freisetzung wird berechnet als Produkt des Wasserspeicherzuflusses (flüssiger Throughfall plus Schmelze) und dem Grad, zu dem die aktuelle Wasserspeicherkapazität laut *smooth1* erreicht ist. Entspricht die aktuelle Wasserspeicherung dem HBV_{96}-Maximalwert, werden 50 % des flüssigen Throughfalls sowie der Schmelze freigesetzt und 50 % von der Schneedecke zurückgehalten. Die Umsetzung gewährleistet, dass wie bei HBV_{96} nur dann Wasser freigesetzt wird, wenn Niederschlag oder Schmelze auftritt, und dass der Glättungsparameter s_1 nicht die Wirkung eines Retentionsparameters zeigt.

$$ IN_t^k = \left({}_{WC}TF_z^k + MELT_z^k\right) \cdot f_{smooth1}\left(WC_z^k - WHC \cdot SP_z^k;\ s_1\right) \qquad 2.95 $$

Für Gletscherzonen ist der Freisetzungsrate der Schneedecke die Gletscherschmelze hinzuzuaddieren (Gleichung 2.22 bzw. 2.96). Die potenzielle Gletschereisschmelzrate errechnet sich äquivalent zur Schneeschmelzrate (Gleichung 2.90) über die Glättungsfunktion *smooth2*. Zur stetigen Reduzierung der Gletscherschmelze aufgrund von Schneebedeckung wird der Wirkungsgrad des Eisanteils der Schneedecke (Gleichung 2.92) herangezogen.

$$IN_z^k = IN_t^k + f_{smooth2}\left(Tc_i^k - TTM;\ s_2\right)\cdot\left(1 - {}^*SP_z^k\right) \qquad 2.96$$

2.3.4 Boden

In den Differenzialgleichungen des Bodenmoduls sind mit den Grenzwerten der Bodenfeuchte, des oberen Grundwassers und des Eisanteils der Schneedecke in Verbindung stehende Unstetigkeiten zu eliminieren.

In HBV_{96} kann die Bodenfeuchte Werte zwischen null und der Feldkapazität annehmen (Gleichung 2.71). Die auf die Feldkapazität bezogene relative Bodenfeuchte bestimmt den Abflussbeiwert (Gleichung 2.65), den potenziellen kapillaren Wiederaufstieg (Gleichung 2.66) und die contributing area (Gleichung 2.73). Entsprechend können beide Grenzwerte nicht unabhängig voneinander durch Glättungsfunktionen ersetzt werden, wie z. B. beim Interzeptionsspeicher der Fall, dessen Grenzwerte lediglich festlegen, ob einzelne Flüsse auftreten oder nicht. Stattdessen ist der auf die Feldkapazität bezogene Wirkungsgrad der Bodenfeuchte ${}^*SM$ (vgl. Gleichung 2.92), durch Normierung der oberen Grenze auf den Wert eins und anschließende Verwendung von *smooth3* zu ermitteln (Gleichung 2.97):

$${}^*SM_z^k = f_{smooth3}\left(\frac{SM_z^k}{FC};\ s_3\right) \qquad 2.97$$

mit: *SM Wirkungsgrad der Bodenfeuchte [mm]

Der für Gleichung 2.97 spezifische Glättungsparameter berechnet sich nach Gleichung 2.98:

$$s_3 = p\left(\frac{s_W}{FC}\right) \qquad 2.98$$

Gleichung 2.65 und 2.66 zur Berechnung von potenziellem Abfluss und potenziellem kapillaren Aufstieg werden ersetzt durch Gleichung 2.99 und 2.100, in denen der Wirkungsgrad der Bodenfeuchte zum Tragen kommt.

$$R_z^k = IN_z^k \cdot \left({}^*SM_z^k\right)^{BETA} \qquad 2.99$$

$${}_{pot}CF_z^k = CFLUX \cdot \left(1 - {}^*SM_z^k\right) \qquad 2.100$$

Gleichung 2.67 zur Berechnung des tatsächlichen kapillaren Aufstiegs wird ersetzt durch Glei-

chung 2.101, in welcher der geglättete potenzieller Aufstieg mit dem Wirkungsgrad des oberen Grundwassers multipliziert wird.

$$CF_z^k = {}_{pot}CF_z^k \cdot f_{smooth1}\left(UZ_z;\ s_1\right) \qquad 2.101$$

Gleichung 2.102 definiert die Verdunstung aus dem Bodenkörper und ersetzt die Gleichungen 2.68 und 2.69. Bei Belegung des Parameters *ERED* mit Werten größer null wird zunächst die potenzielle Verdunstung um die priorisierte aktuelle Interzeptionsverdunstung reduziert. Die zweite Reduktion ist dem Feuchtedefizit des Bodens geschuldet. Mittels *smooth3* wird analog zu Gleichung 2.97 ein verdunstungsbezogener Wirkungsgrad der Bodenfeuchte definiert. Allerdings ist der Wirkungsgrad auf denjenigen Feuchtgehalt bezogen, ab dem entsprechend der HBV_{96}-Konzeption die Maximalverdunstung auftritt. Als dritten Reduktionsfaktor postuliert HBV_{96} die Verhinderung des Luftaustausches zwischen Boden und Atmosphäre durch Schnee. Analog zur Reduktion der Gletscherschmelze nach Gleichung 2.96, berechnet sich der Reduktionsfaktor über den Wirkungsgrad des Eisanteils der Schneedecke.

$$EA_z^k = \left(EPc_i^k - ERED \cdot EI_z^k\right) \cdot f_{smooth3}\left(\frac{SM_z^k}{LP \cdot FC};\ s_3\right) \cdot \left(1 - {}^*SP_z^k\right) \qquad 2.102$$

Der für Gleichung 2.102 spezifische Glättungsparameter berechnet sich nach Gleichung 2.103:

$$s_3 = p\left(\frac{s_W}{LP \cdot FC}\right) \qquad 2.103$$

2.3.5 Abflusskonzentration bzw. Grundwasser

Die contributing area berechnet sich entsprechend Gleichung 2.73, allerdings unter Rückgriff auf den in Gleichung 2.97 definierten Wirkungsgrad der relativen Bodenfeuchte, wie folgt (Gleichung 2.104):

$$contriArea_z = \begin{cases} \sum_{k \in Z_S}\left(\frac{A^k}{\sum_{k \in Z_S} A^k} \cdot \left({}^*SM_z^k\right)^{BETA}\right) & falls \quad \left(RESPAREA = 1\right) \cap \left(Z_S \neq \varnothing\right) \\ 1 & falls \quad \left(RESPAREA = 0\right) \cup \left(Z_S = \varnothing\right) \end{cases} \qquad 2.104$$

Gleichung 2.74 zur Berechnung der Perkolation vom oberflächennahen zum tiefen Grundwasser wird durch Gleichung 2.105 ersetzt. Die Reduktion der maximal möglichen Perkolation erfolgt durch Multiplikation der contributing area sowie des Wirkungsgrades des oberen Grundwassers.

$$PERC_z = PERCMAX \cdot contriArea_z \cdot f_{smooth1}\left(UZ_z;\ s_1\right) \qquad 2.105$$

Die Berechnung des Direktabflusses ist identisch zu Gleichung 2.75 und wird lediglich der

Vollständigkeit halber mit aufgeführt (Gleichung 2.106):

$$Q0_z = K \cdot \left(\frac{UZ_z}{contriArea_z} \right)^{ALPHA+1} \qquad 2.106$$

Die Berechnung des Basisabflusses nach Gleichung 2.75 wird ersetzt durch Gleichung 2.107. Für das untere Grundwasser können unabhängig von der Belegung von s_1 aufgrund der Seeverdunstung beliebig extreme Negativwerte zustande kommen. Dies schränkt die Anwendbarkeit von Gleichung 2.107 aber nicht ein, da *smooth1* auch extreme Unterschreitungen von null in das Intervall von null bis eins projiziert.

$$Q1_z = K4 \cdot f_{smooth1}\left(LZ_z;\ s_1\right) \qquad 2.107$$

Nach HBV_{96} entspricht die Seeverdunstung der potenziellen Verdunstung, wird aber durch Seeeis bzw. durch Temperaturen unterhalb des Grenzwertes *TTICE* unterdrückt (Gleichung 2.79). *smooth1* dient in Gleichung 2.108 der Ermittlung eines stetigen Aktivierungsgrades der Verdunstung im Bereich von *TTICE*.

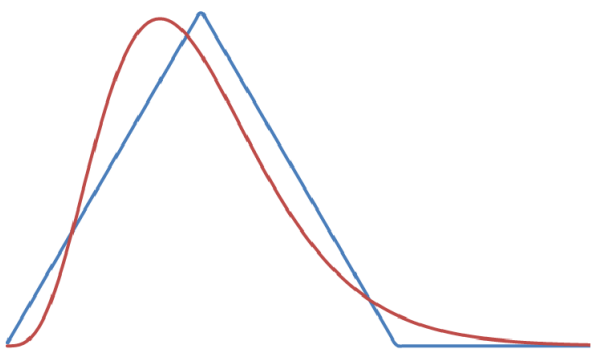
$$EA_i^k = EPc_i^k \cdot f_{smooth1}\left(Tc_i^k - TTICE;\ s_1\right) \qquad 2.108$$

Gleichung 2.109 ist mit Gleichung 2.79 beinahe identisch. Die einzige Änderung betrifft den Verdunstungsterm, in welchem die Indexmenge Z_l sämtliche internal lake-Zonen umfasst und nicht nur diejenigen, deren Lufttemperatur oberhalb von *TTICE* liegt.

$$LZ_{\tau+\Delta} = \begin{cases} LZ_\tau + \Delta \cdot \left(\dfrac{\sum_{k \in Z_L} A^k}{\sum_{k \in Z} A^k} \cdot PERC_{\tau \to \tau+\Delta} - Q1_{\tau \to \tau+\Delta} + \ldots \right. \\ \qquad \left. \ldots \sum_{k \in Z_l} \left(\dfrac{A^k}{\sum_{k \in Z} A^k} \cdot R^k_{\tau \to \tau+\Delta} \right) + \sum_{k \in Z_l} \left(\dfrac{A^k}{\sum_{k \in Z} A^k} \cdot \left(Pc_i^k - EA_i^k \right) \right) \right) \end{cases} \qquad 2.109$$

In HBV_{exp} wird die weitere Abflusskonzentration mit einer Speicherkaskade berechnet. Diese ersetzt den Dreiecks-Unit-Hydrograph von HBV_{96}, womit die Gleichungen 2.43 und 2.44 hinfällig sind.

Die Formulierung der Speicherkaskade in der Zustandsraum-Formulierung erfordert eine ganzzahlige Speicheranzahl. Die Einschränkung ließe sich zwar durch die Verwendung der Impulsantwort der linearen Speicherkaskade umgehen, welche unter Rückgriff auf die Gammafunktion auch nicht ganzzahlige



Speicheranzahlen zulässt. Der Ansatz der Impulsantwort zieht allerdings Faltungsoperationen nach sich, die aus dem oben genannten Grund umgangen werden sollen. Um keine Unstetigkeit in den Raum der Kalibrierparameter einzuführen und um die Anzahl der Kalibrierparameter konstant zu halten, wird die Speicheranzahl fix gesetzt. Hierfür wird der Wert fünf gewählt, da sich die Systemreaktionen von linearer Speicherkaskade und Dreiecks-Unit-Hydrograph bei dieser Speicheranzahl stark ähneln, was Abbildung 2.15 veranschaulicht. Formaler ausgedrückt ist – bei identischer, gegenüber der Zeitschrittweite nicht zu kleiner Schwerpunktlaufzeit – die Summe der quadrierten Differenzen beider Systemreaktionen auf einen Δt-Impuls bei einer Speicheranzahl von fünf minimal.

Abbildung 2.15: Speicherkaskade vs. UH

Der Zufluss zur Speicherkaskade entsprechend Gleichung 2.110 berechnet sich analog zu Gleichung 2.44.

$$QSC_z^0 = \frac{\sum_{k \in Z_L} A^k}{\sum_{k \in Z} A^k} \cdot Q0_z + Q1_z \qquad 2.110$$

mit: QSC^0 Zufluss der Speicherkaskade [mm/h]

Die Ausflüsse der fünf Einzelspeicher hängen zunächst von der Kaskaden-Schwerpunktlaufzeit *KSC* ab, die für jeden Einzelspeicher um das Fünffache zu verringern ist. Des Weiteren vom Inhalt des jeweiligen Speichers, wobei durch die Verwendung der Glättungsfunktion *smooth2* zumindest im unteren Wertebereich kein linearer Zusammengang besteht. Hintergrund ist die Eliminierung des unteren Grenzwertes null der Einzelspeicher. Dieses erleichtert die numerische Integration, kann bei sehr kleinen Wertbelegungen von *KSC* aber zu einer erhöhten Schwerpunktlaufzeit führen. Aus dem genannten Grund und zur Vermeidung numerischer Instabilitäten ist für *KSC* als Minimalwert eine Stunde angesetzt. s_2 ist mit dem fixen Wert 0,02 belegt.

$$QSC_z^{1:5} = \frac{5}{KSC} \cdot f_{smooth2}\left(SSC_z^{1:5};\ s_2\right) \qquad 2.111$$

mit:

$QSC^{1:5}$	Ausflüsse der fünf Einzelspeicher	[mm/h]
$SSC^{1:5}$	Inhalte der fünf Einzelspeicher	[mm]
KSC	Schwerpunktlaufzeit	[1/h]

Der Ausfluss des fünften Speichers ist gleichzeitig der Gesamtausfluss eines Teilgebietes vor eventuellen Entnahmen bzw. der Zufluss in einen outlet lake (Gleichung 2.112):

$$Q_z = QSC_z^5 \qquad 2.112$$

Die Aktualisierung der einzelnen Speicher ergibt sich entsprechend ihrer Zu- und Abflüsse, was die um den Wert eins verschobenen Indices von *QSC* andeuten (Gleichung 2.113):

$$SSC_{\tau+\Delta}^{1:5} = SSC_{\tau}^{1:5} + \Delta \cdot \left(QSC_{\tau \to \tau+\Delta}^{0:4} - QSC_{\tau \to \tau+\Delta}^{1:5} \right) \qquad 2.113$$

Im die Kompatibilität zur HydPy-Implementierung von HBV_{96} zu gewährleisten, wird der HBV_{exp}-Parameter *KSC* automatisch aus dem HBV_{96}-Parameter *MAXBAZ* abgeleitet, falls nur letzterer vom Benutzer in der entsprechenden Modell-Initialisierungsdatei hinterlegt ist (Abschnitt 1.9.3). Da *KSC* die Schwerpunktlaufzeit und *MAXBAZ* die Basis des symmetrischen Dreiecks-Unit-Hydrographs darstellt, gilt (Gleichung 2.114):

$$KSC = \frac{MAXBAZ}{2} \qquad 2.114$$

2.3.6 Vergleich HBV_{exp} mit HBV_{96}

Die folgenden Vergleichsrechnungen wurden unter den in Abschnitt 2.2.5 genannten Randbedingungen durchgeführt.

Abbildung 2.16 ist eine Dauerlinien-Darstellung des Grades der Übereinstimmung der verschiedenen Modellvarianten, der analog zu Tabelle 2.8 mittels Nash-Sutcliffe-Koeffizienten quantifiziert ist. Verglichen werden die Abflüsse von HBV_{96} und HBV_{exp} an allen Teilgebietsauslässen. Um nur den Effekt der Eliminierung der Unstetigkeiten zu analysieren, dienen die Simulationsergebnisse von HBV_{96} entsprechend der Zustandsraum-Implementierung als Referenz. Zur Veranschaulichung der Wirkung verschiedener Glättungsgrade wurde die HBV_{exp}-Simulation mit 0,1 und mit 1,0 als Wert für beide Meta-Glättungsparameter durchgeführt. In beiden Fällen sind die Abweichungen gering, was zeigt, dass das „makroskopische“ Systemverhalten von HBV_{96} weitgehend erhalten bleibt. Wie zu erwarten wachsen die Abweichungen zwischen HBV_{96} und HBV_{exp} mit Zunahme des Glättungsgrades.

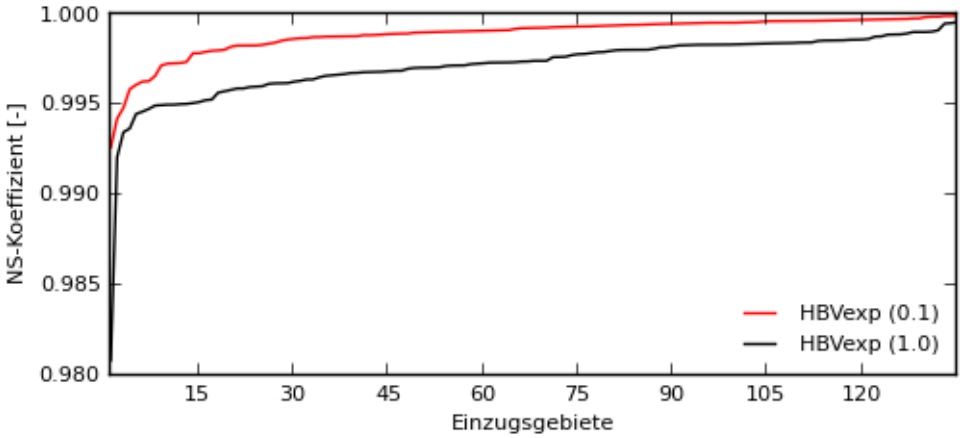


Abbildung 2.16: HBV_{exp} vs HBV_{96}, alle Teilgebiete

Tabelle 2.10: HBV_{96} vs. HBV_{exp}, Neckar 1

Flussgröße [-]	s_m = 0,1 [mm bzw. °C]	s_m = 1,0 [mm bzw. °C]
Pc	1.00000	0.99999
TF	0.99833	0.97244
EI	0.95144	0.71488
IN	0.99757	0.96906
R	0.99678	0.97098
EA	0.99896	0.99619
PERC	0.97850	0.91359
Q0	0.99867	0.99455
Q1	0.99910	0.99792
Q	0.99852	0.99446

Tabelle 2.10 zeigt analog zu Tabelle 2.9 exemplarisch für das Teilgebiet Neckar 1, in welchem Ausmaß das Systemverhalten einzelner Modellkomponenten durch die Eliminierung der Unstetigkeiten in den entsprechenden Differenzialgleichungen modifiziert wird. Am stärksten betroffen ist das Interzeptionsmodul. Grund hierfür ist die Relation der Speichergröße *ICMAX* zum Glättungsparameter s_W. *ICMAX* beträgt 1,0 mm für Field- und 1,5 mm für Forst-Zonen. Insbesondere bei Belegung von s_W mit dem Wert 1,0 mm bewegt sich der Glättungsbereich in derselben Größenordnung wie der ursprüngliche Zustandsbereich, was zu einer „makroskopischen" Änderung des Systemverhaltens dieser Modellkomponente führt. Warum die Verringerung des Nash-Sutcliffe-Koeffizienten für die Interzeptionsverdunstung *EI* deutlich stärker als für den Throughfall *TF* ausfällt, veranschaulichen Abbildung 2.17 und Abbildung 2.18.

Bei einem s_W von 0,1 mm ist der Unterschied gegenüber dem mit HBV_{96} berechneten Throughfall gering. Bei einem s_W von 1,0 mm zeigen die HBV_{exp}-Ergebnisse dagegen eine nennenswerte Verflachung des „makroskopischen" Throughfall-Verlaufes. Nichtsdestotrotz steht die Variabilität des Througfalls weiterhin primär mit der des eingehenden Niederschlages in Verbindung, weshalb das ursprüngliche zeitliche Muster bestehen bleibt und sich der Nash-Sutcliffe-Koeffizient nur wenig verringert.

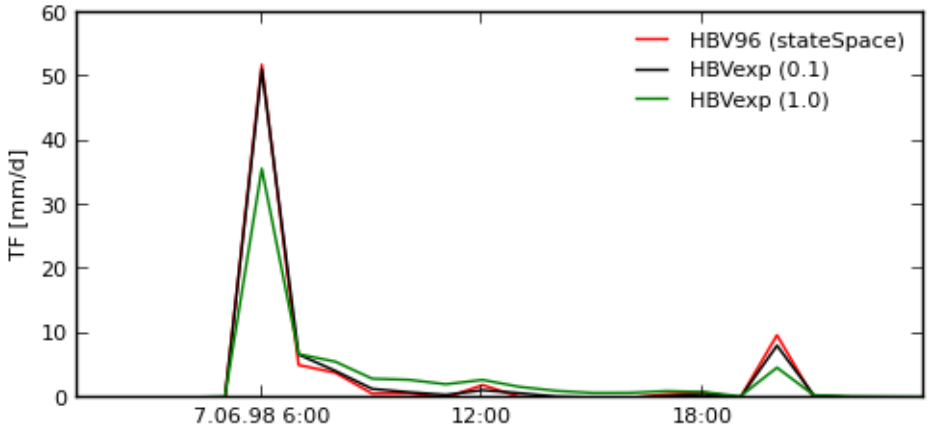


Abbildung 2.17: HBV_{exp} vs. HBV_{96}, Throughfall, Neckar 1

Kleine makroskopische Änderungen des Verlaufes der Interzeptionsverdunstung zeigen sich bereits bei einem s_W von 0,1 mm. Aufgrund der eliminierten scharfen Grenzen null und *ICMAX* und des – im Gegensatz zum Niederschlag – nicht intermittierenden Verhaltens der potenziellen

Verdunstung beginnt und endet die Interzeptionsverdunstung gewissermaßen kontinuierlich. In den Phasen mit konstant gefülltem oder leerem Speicher wirkt sich dies nur geringfügig aus. Bei einem s_W von 1,0 mm greifen die Übergangsphasen aber derart weit in die trockenen und feuchten Phasen aus, dass sich das zeitliche Muster der Interzeptionsverdunstung markant ändert, woraus die starke Verringerung des Nash-Sutcliffe-Koeffizienten resultiert.

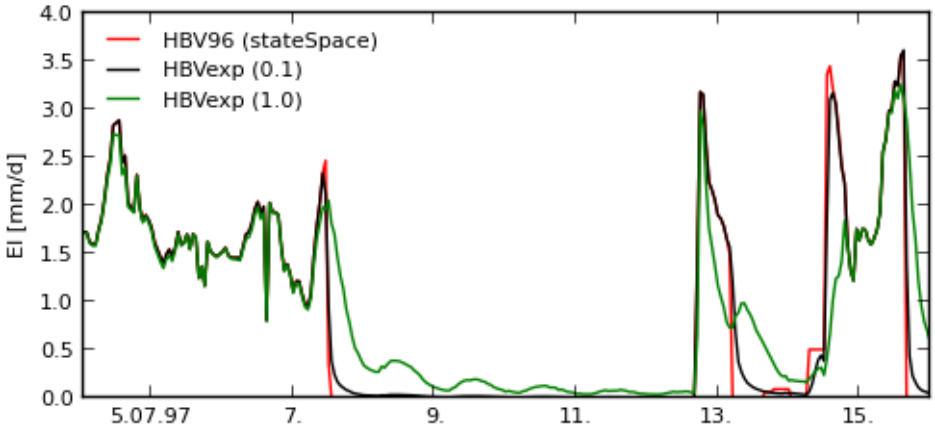


Abbildung 2.18: HBV_{exp} vs. HBV_{96}, Interzeptionsverdunstung, Neckar 1

Abbildung 2.19 veranschaulicht die starke Übereinstimmung der Gesamtreaktion von HBV_{96} und HBV_{exp} bei geringem Glättungsgrad. Beim größeren Glättungsgrad zeigt die mit HBV_{exp} simulierte Abflussganglinie eine leichte Verflachung und liegt – aufgrund der erhöhten Interzeptionsverdunstung – überwiegend unterhalb der HBV_{96}-Ganglinie.

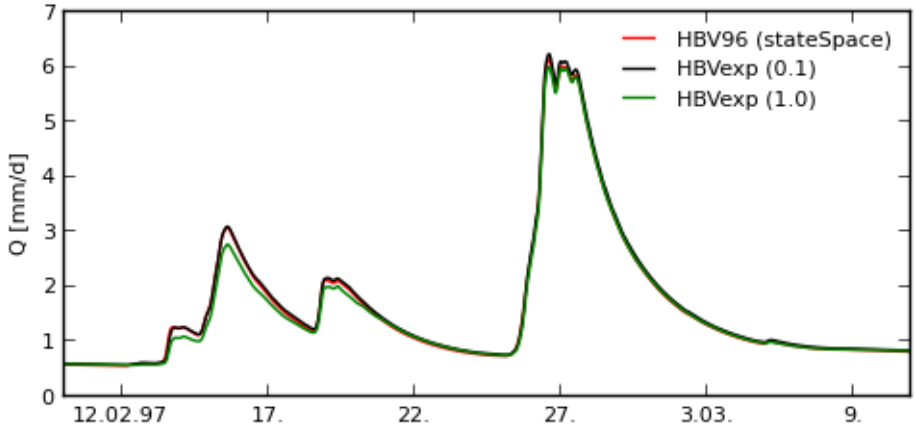


Abbildung 2.19: HBV_{exp} vs. HBV_{96}, Abfluss, Neckar 1

Literaturverzeichnis

Clark, M. P. & Kavetski, D. (2010): Ancient numerical daemons of conceptual hydrological modeling: 1. Fidelity and efficiency of time stepping schemes. In: *Water Resour. Res.* 46 (10).

Cython core developers (Hrsg.) (2012): Cython: C-Extensions for Python. Online verfügbar unter http://cython.org/, zuletzt aktualisiert am 26.09.2012, zuletzt geprüft am 19.10.2012.

Dahmen, W. & Reusken, A. (2006): Numerik für Ingenieure und Naturwissenschaftler. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg (Springer-Lehrbuch). Online verfügbar unter http://gbv.eblib.com/patron/FullRecord.aspx?p=1156426.

Ernesti, J. & Kaiser, P. (2008): Python. Das umfassende Handbuch. 1. Aufl., 1., korrig. Nachdr. Bonn: Galileo Press (Galileo Computing). Online verfügbar unter http://deposit.d-nb.de/cgi-bin/dokserv?id=3001841&prov=M&dok_var=1&dok_ext=htm.

ESRI Deutschland (Hrsg.) (2012): ArcGIS. Online verfügbar unter http://www.esri-germany.de/products/arcgis/index.html, zuletzt aktualisiert am 20.09.2012, zuletzt geprüft am 19.10.2012.

Kavetski, D. & Kuczera, G. (2007): Model smoothing strategies to remove microscale discontinuities and spurious secondary optima in objective functions in hydrological calibration. In: *Water Resour. Res.* 43 (3).

Lindström, G., Johansson, B., Persson, M., Gardelin, M. & Bergström, S. (1997): Development and test of the distributed HBV-96 hydrological model. In: *Journal of Hydrology* 201 (1-4), S. 272–288.

MathWorks Deutschland (Hrsg.): MATLAB - The Language of Technical Computing. Online verfügbar unter http://www.mathworks.de/products/matlab/, zuletzt geprüft am 19.10.2012.

McCarthy, G. (1939): The Unit Hydrograph and Flood Routing. Providence: Army Engineer District.

Numpy developers (Hrsg.) (2012a): Numpy license — Numpy. Online verfügbar unter http://numpy.scipy.org/license.html#license, zuletzt aktualisiert am 11.05.2012, zuletzt geprüft am 19.10.2012.

Numpy developers (Hrsg.) (2012b): Scientific Computing Tools For Python — Numpy. Online verfügbar unter http://numpy.scipy.org/, zuletzt aktualisiert am 11.05.2012, zuletzt geprüft am 19.10.2012.

Python Software Foundation (Hrsg.) (2012a): History and License — Python v2.7.3 documentation. Online verfügbar unter http://docs.python.org/license.html, zuletzt aktualisiert am 19.10.2012, zuletzt geprüft am 19.10.2012.

Python Software Foundation (Hrsg.) (2012b): Python Programming Language – Official Website. Online verfügbar unter http://www.python.org/, zuletzt aktualisiert am 19.10.2012, zuletzt geprüft am 19.10.2012.

Pythonxy developers (Hrsg.) (2012): pythonxy - Scientific-oriented Python Distribution based on Qt and Spyder - Google Project Hosting. Online verfügbar unter http://code.google.com/p/pythonxy/, zuletzt geprüft am 19.10.2012.

SMHI (2011): Manual IHMS - Integrated Hydrological Modelling System. Version 6.2.

SMHI (Hrsg.) (2009): Improvement HBV model Rhine in FEWS. Final report. Unter Mitarbeit von G. Berglöv, J. German, H. Gustavsson, U. Harbman und B. Johansson (Hydrology, 112).

The Apache Software Foundation (Hrsg.) (2012a): Apache Subversion. Online verfügbar unter http://subversion.apache.org/, zuletzt geprüft am 19.10.2012.

The Apache Software Foundation (Hrsg.) (2012b): Apache License, Version 2.0. Online verfügbar unter http://www.apache.org/licenses/LICENSE-2.0.html, zuletzt aktualisiert am 17.10.2012, zuletzt geprüft am 19.10.2012.

The R Foundation (Hrsg.) (2006): The R Project for Statistical Computing. Online verfügbar unter http://www.r-project.org/, zuletzt aktualisiert am 21.04.2006, zuletzt geprüft am 19.10.2012.

The TortoiseSVN team (Hrsg.) (2012): TortoiseSVN. Online verfügbar unter http://tortoisesvn.net/, zuletzt aktualisiert am 09.10.2012, zuletzt geprüft am 19.10.2012.

Vaingast, S. (2009): Beginning Python Visualization. Crafting Visual Transformation Scripts. Berkeley, CA: Apress. Online verfügbar unter http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10288823.