Runge-Kutta (RK) Generator

Alexander Schaap

December 19, 2017

1 Revision History

The latest version can be found at https://github.com/aschaap/cas741.

Date	Version	Notes
October 20, 2017	1.0	Initial version
December 17, 2017	1.1	Addressed feedback
December 19, 2017	1.2	Final version

2 Reference Material

This section records information for easy reference.

2.1 Table of Units

Since the application of this software is dependent on the program using the functions provided by this program family, no physical units are used throughout this document.

2.2 Table of Symbols

The table that follows summarizes the symbols used in this document. As stated above, no physical units are associated with any of these. The choice of symbols was made to be consistent with the literature describing ODEs.[1] The symbols are listed in alphabetical order.

symbol	description
$\mathbb{C}$	Complex numbers
$\mathbf{f}$	ODE for which to solve
h	step size
$\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3, \mathbf{k}_4$	intermediate variables in the RK4 method, similar to $\mathbf{Y}_1$ [I don't know what the subscript notation means here. Do you have 4 vectors of k values or one vector with 4 values? —SS] [I meant 4 vectors, but I've tried to make it less ambiguous. —AS]
$\mathbb{R}$	Real numbers
t	an independent variable, commonly time
t_0	beginning of interval
t_N	end of interval
t_k	any t on the interval $[t_0..t_N]$
$\mathbf{x}(\mathbf{t})$	a function of t
$\mathbf{x}'$	derivative of $\mathbf{x}(\mathbf{t})$ with respect to t [add: with respect to t —SS][Done. —AS]
$\mathbf{x}_0$	initial values for $\mathbf{f}$
$\mathbf{x}_k$	(numerical approximation of) a point on the solution for $\mathbf{f}$
$\mathbf{Y}_1$	intermediate variables in the RK2 method, similar to $\mathbf{k}_1..\mathbf{k}_4$

[1] Specifically, Corless and Fillion (2013).

2.3 Abbreviations and Acronyms

symbol	description
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
ODE	Ordinary Differential Equation
PS	Physical System Description
R	Requirement
RK	Runge-Kutta
RK2	Second order Runge-Kutta method
RK4	Fourth order Runge-Kutta method
RK Generator	Family of programs based on the RK2 / RK4 method(s)
SRS	Software Requirements Specification
STEM	Science, Technology, Engineering & Mathematics
T	Theoretical Model

Contents

3 Introduction

This document provides an overview of the commonality analysis (CA) for the RK Generator program family. Members of program family are produced by a code generator. Generated members provide numerical approximations for given ordinary differential equations (ODEs) using Runge-Kutta (RK) methods. Most of the calculations happens during code generation, producing a different family member for each given combination of RK method, ODE, interval, step size, and initial values. The current section describes the purpose of this document, the scope of this family, the organization of the remainder of the document and the characteristics of the intended reader.

3.1 Purpose of Document

The main purpose of this document is to provide sufficient information to understand what the RK Generator is expected to do. The goals and theoretical models used in the RK Generator are provided, as are assumptions and unambiguous definitions. This document should ultimately aid the development of the software by specifying and constraining the design, and by providing a starting point for the creation of test cases. It should also give developers maintaining and/or improving the software a better idea of what it should and should not do, though improvement could require them to update this document. For users (i.e. developers looking to incorporate this into their own software), this document details the uses and limitations of the RK Generator. Characteristics of Intended Reader covers the intended audience in more detail.

3.2 Scope of the Family

The responsibility of the family is solving the provided first order ODEs (in explicit form) for given intervals, initial values and uniform step sizes using the specified RK method. The user will have to ensure the RK method specified is appropriate for their particular problem(s), and they will have to ensure appropriate and correct input is provided (see Data Definitions). Additionally, the user will also have to process the output appropriately; namely use the generated function that will take inputs within the given interval and provide approximate solutions for them. [Your scope should specify which kind of ODEs you solve. Your software solves systems of first order ODEs. You do not solve higher order ODEs, presumably because they can be recast as systems of first order ODEs. In addition, the functional form of your ODE can be given explicitly, as opposed to being in implicit form. —SS] [I've added this. —AS]

3.3 Characteristics of Intended Reader

The reader is expected to have some undergraduate background in a math-heavy science or engineering, as well as a graduate understanding of code generation. Ideally, they have been exposed to some undergraduate calculus courses that covered numerical methods for ODE's.

Programming courses would also help, especially a graduate understanding of functional programming and metaprogramming. [For the characteristics of intended reader try to be more specific about the education. What degree? What course areas? What level? —SS] [I've tried to add some detail. —AS]

3.4 Organization of Document

The organization of this document follows the template for a commonality analysis (CA) [The citations are enough; you don't need to mention me separately —SS] [Removed your name. —AS] adapted from Lai (2004), Smith and Lai (2005), Smith (2006) and Smith et al. (2007).

The structure of this document is essentially top-down, starting with a birds-eye view with detail added as one proceeds through it. A general system description is provided first, consisting of context, user characteristics and system constraints, followed by commonalities and variabilities. The commonalities section includes some background information, terminology and definitions, data definitions, goals and theoretical models. Variabilities include assumptions, different ways of calculating the result, and how output may (not) differ between variants. Traceability matrices and references are listed at the end.

4 General System Description

This section identifies the interfaces between the system and its environment, describes the potential user characteristics and lists the potential system constraints.

4.1 Potential System Contexts

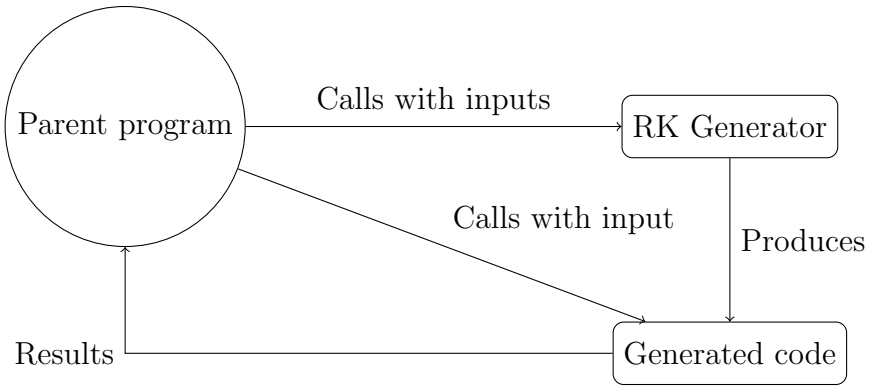

Figure 1: System Context

Figure 1 shows the system context. A circle represents than external entity outside the software, which is the "user", or parent program making use of the generator as a library.

A rectangle is used to represent the generator as well as the RK Generator family members it can generate.

Using the library is as simple as using an **open** statement, though even this is not strictly necessary. If it is used, it will be compiled into the parent program. Ideally, the parent program knows the ODE(s) and other inputs at compile time, and provides them to the generator when calling it to generate family members. The parent program subsequently uses each generated family member many times at run-time for different points within the interval, to maximize the benefit of code generation.

Alternatively, the ODE(s) or other inputs are not known at compile time; the generator can still be called at run-time, but hopefully the parent program can offset the incurred cost of this on-the-fly generation by running the optimized generated code enough times.

A breakdown of responsibilities of both the user and the RK Generator is provided below:

- User Responsibilities:
  - Provide appropriate input to the system
  - Ensure assumptions on the input are met (see subsection 6.1)
  - Correctly process the resulting output function
- RK Generator Responsibilities:
  - Calculate the correct output for the given input

4.2 Potential User Characteristics

The most common user of RK Generator will be other programs. However, one or more programmers are needed to write the code that calls the functions provided by this family. These programmers therefore should have an understanding of ODEs, the accuracy of numerical approximation (using RK methods) and OCaml programming. Knowledge of MetaOCaml should not be necessary if the program family is able to hide its usage thereof sufficiently.

4.3 Potential System Constraints

The responsibility of generating family members in a type-safe way restricts the system to the MetaOCaml extension of the OCaml programming language. Code generation errors can be tricky (think C++ templates), so using a type-safe language ensures that if it compiles, it most likely works as expected. This also helps avoid more subtle errors that could easily go unnoticed. Alternatives tend to make less guarantees about the generated code.

5 Commonalities

This section contains the commonalities between the RK Generator family members. This section starts with succinct background information about the problem, followed by terminology and definitions, data definitions, goal statements and ending with theoretical models.

5.1 Background Overview

There are various numerical methods for approximating ordinary differential equation (ODE) given initial values. This problem is called an initial value problem (IVP, see T1). A well-known way to solve these problems is through Runge-Kutta (RK) methods.

To explain RK methods, it is easiest to start with the Euler method. Euler's method consists of simply taking the slope of the current point, and extrapolating that for a distance of step size h on the x-axis to create a new point. Since the slope of the point changes, this method creates an approximation that will always seem to "lag behind" the actual ODE, but with a small enough step size, it should be pretty close. For larger intervals, you'll see that the error appears to accumulate. See Figure 2 as a simple illustration.

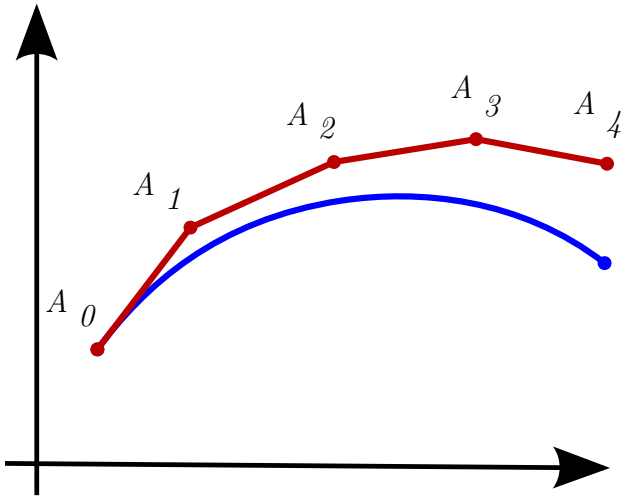

Figure 2: Approximation obtained using Euler's method versus the actual solution

Using Euler's method as a starting point, one can see how accuracy could be improved by taking the slopes at both the current point as well as the current point incremented by step size h, and using the average of those two slopes as the slope with which to extrapolate to approximate the point that is step size h away from the current point. This is the second order Runge-Kutta (RK2) method (for the mathematical definition, see IM2).

Runge-Kutta methods are a collection of methods that use slopes of multiple, sometimes intermediate, points (found through successive extrapolation) to create more accurate approximations. The slopes are also assigned potentially different weights when finding the average. The more points one uses (the higher the order of the RK method), the smaller the error should be (the error will always be within $h^{(order\ of\ used\ RK\ method\ +\ 1)}$, which becomes smaller quite fast for small step sizes).

5.2 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly under-

stand the requirements:

- Code generation: producing code programmatically.
- Compile-time: the time during which a program is compiled.
- Continuous function: a function that evaluates to a value that is not infinity or undefined for any input.
- Initial values (initial condition): a starting point from which the rest of the approximation can be calculated. (See DD3.)
- Initial value problem (IVP): an ODE along with initial values for which a numerical approximation must be found.
- Interval: interval inside which the ODE needs to be approximated. (See DD2.)
- MetaOCaml: an extension to the OCaml programming language which adds staging annotations that enable MSP.
- Metaprogramming: programming software that takes in and/or produces programs.
- Multi-stage programming (MSP): a form of metaprogramming where the execution of specific code fragments is delayed; these fragments are then combined into larger fragments and ultimately executed.
- Numerical method: a specific way to numerically approximate solutions to a particular problem (rather than symbolic manipulations).
- OCaml: A multi-paradigm programming language (imperative & functional) with a syntax that will appear somewhat unusual to most.
- Ordinary differential equation (ODE): an equation that involves some ordinary (rather than partial) derivatives of a function. The goal is to find the function that satisfies the equation, and this is not always easily achievable through integration. (See DD1.)
- Runge-Kutta methods: a set of methods that approximate ODEs in IVPs for a given interval and step size.
- Run-time: the time during which a program is run.
- Step size: the distance between any two points for which to find approximations; it essentially determines how many points there will be once the interval is known. Once these points have been approximated, one could interpolate so that the whole interval is solved for. (See also DD4.)
- Stiff ODE: an ODE for which certain numerical methods are numerically unstable unless the step size is extremely small. There is no exact definition.

- Type-safe: Applicable to programming languages; property that follows from such a language not allowing conversions/reductions of valid (adherent to the language's type system) programs to a state where the type system is violated, nor from where no further reduction is possible. So all valid programs can evaluate to a value in the language's set of evaluation rules. A simple example: whether the type system allows for integers to be used as floating point number.

5.3 Data Definitions

This section collects and defines all the data needed to build the instance models. The dimension of each quantity is also given.

Number	DD1
Label	**Ordinary Differential Equation (ODE)**
Symbol	$\mathbf{f}$
Units	N/A
Equation	$\mathbf{x}' = \mathbf{f}(t, \mathbf{x}(t))$
Description	$\mathbf{f} : \mathbb{R} \times \mathbb{C}^n \to \mathbb{C}^n$ is the equation for which we ultimately want to find a numerical approximation. [I don't think this is true. You know f and you are trying to find x. —SS]
Sources	Corless and Fillion (2013)
Ref. By	IM3, IM2, T1, A2, A3

Number	DD2
Label	**Interval**
Symbol	t_0, t_N
Units	N/A
Equation	N/A
Description	$[t_0, t_N] : \mathbb{R} \times \mathbb{R}$ is the interval for which to solve the ODE (see DD1). t_0 represents the beginning of the interval, t_N the end.
Sources	Corless and Fillion (2013)
Ref. By	IM3, IM2, A2, A4

Number	DD3
Label	**Initial values**
Symbol	$\mathbf{x}_0$
Units	N/A
Equation	$\mathbf{x}(t_0) = \mathbf{x}_0$
Description	$\mathbf{x}_0 : \mathbb{C}^n$ are initial values for solving ODE (see DD1).
Sources	Corless and Fillion (2013)
Ref. By	DD4, IM3, IM2, T1, A3

Number	DD4
Label	**Step size**
Symbol	h
Units	N/A
Equation	$t_{k+1} - t_k = h$
Description	$h : \mathbb{R}$ is the distance between the points within interval $[t_0, t_N]$ for which to find approximations.
Sources	Corless and Fillion (2013)
Ref. By	IM3, IM2

5.4 Goal Statements

Given an RK method specification, the specified ODE (DD1; non-stiff and continuous, as per A2), an interval (DD2) and the desired step size (DD4), as well as an initial condition (DD3), the goal statement is:

GS1: Given Calculate solutions for and generate (in a type-safe manner) a function that uses these solutions to solve for specific points (multiples of the step size) within the provided interval. Type-safe code generation means that the code snippets or fragments will have types, rather than being strings for example. If done properly, this allows only valid code to be generated.

In phase 2 (starting in January 2018), spline interpolation (between the points that fall on multiples of the step size) will be added. This will allow the generated function

to solve for any point within the interval, rather than only specific ones.

[Why are you mentioning a spline? In addition to solving the ode at the specified points, are you interpolating between the points using a spline? If so, you should add information defining a spline, and you'll need to be more specific on what spline you are using in terms of order and continuity conditions at the knots. —SS] [Unfortunately, it seems I was too ambitious. Splines have been moved to phase 2 that starts in January. —AS]

5.5 Theoretical Models

This section focuses on the problem that RK Generator is designed to solve.

Number	T1
Label	**Initial value problem (IVP)**
Input	• $\mathbf{f} : \mathbb{R} \times \mathbb{C}^n \to \mathbb{C}^n$ is the function describing the vector field (see DD1) • $t_0, t_N : \mathbb{R} \times \mathbb{R}$ the bounds of the interval within which to solve (see DD2) • $\mathbf{x}_0 : \mathbb{C}^n$ is the initial condition (see DD3) • $h : \mathbb{R}$ the step size (see DD4)
Output	• anonymous function : $\mathbb{R} \to \mathbb{R}^n$ This function takes in a value t for which to return an approximation of the given ODE at that point.
Description	The inputs of an initial value problem are given above. The issue is that many IVPs are difficult to solve manually (or programmatically) and the correct solutions are often unknown. Numerical methods such as RK methods are close enough to be used in most applications.
Source	Corless and Fillion (2013) p. 510, 513
Ref. By	IM3, IM2

[It would be better if you used the instance model template for your theoretical model. In particular, a clear separation between the inputs and the ouputs would be nice. —SS] [I separated the Equation row into Input and Output. —AS]

6 Variabilities

Assumptions on all variations of the program family are covered first. Variability in RK method in the RK Generator program family are represented by instance models in Calculation. Every unique input will result in slightly different code being generated, which could in turn produce different output compared to other family members. This allows for an infinite program family depending on one's definition of program family[2]. After instance models, the lack of variability in output is covered.

6.1 Assumptions

A1: Only the second (IM3) and fourth (IM2) order RK methods will be used. [This is a variability, but it isn't an assumption. —SS] [I're changed this to reflect which instance models are available as per Geneva's suggestion. —AS]

A2: The ODE (DD1) provided will be continuous (particularly on the interval of computation (DD2)) and non-stiff (otherwise the user accepts that the results will be inaccurate).

[The assumptions don't need to repeat things you have alread said. —SS] [Removed this assumption. —AS]

A3: Initial value (DD3) vector size is expected to match the ones produced by the ODE (DD1).

A4: The interval's bounds (DD2) satisfy $t_0 < t_N$ (both real numbers) and $t_N - t_0 = n \times h$. This will make adding interpolation in phase 2 easier. [In general with floating point arithmetic this won't necessarily be true. In practice if h is added to the time for each time step, you won't end exactly at t_N. Do you really need this assumption to be true? —SS] [No, but it is useful for when interpolation is added. —AS]

[You can give this through the type of h; you don't need an assumption for this information. —SS] [I've removed this assumption as well. —AS]

[It is okay to have a short list of assumptions. It feels like you might have been trying to "pad" the list. What about an assumption that f is continuous on the interval of computation? —SS] [I've added this to A2 —AS]

[2]Currently Dr. Smith and Dr. Carette agree they should discuss this definition to compare their overlapping but slightly differing viewpoints. [By default, LaTeX puts two spaces after a period, since it assumes that the period ends a sentence. To override this, use x. y or x. y. The first puts one space, the second option also does one space, with the added constraint that a line break cannot happen at the space. —SS]

6.2 Calculation

The family can utilize either of the instance models listed below for any of its members. As stated in IM1, the functions ϕ and $\mathbf{f}(t_k, \mathbf{x}_k)$ constitute two variabilities, as different definitions of either of these will yield different program family members.

[To make this clear that you have a family of methods, I would introduce an instance model for one-step initial value problem solvers. The equation would be

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\phi(t_k, \mathbf{x}_k, h_k)$$

The only difference for your family members is how ϕ is defined. This approach would let you easily add new family members in the specification, simply by giving new values for ϕ. —SS] [I've done this; I hope I'm still using the template constructs correctly. I'm considering the possibility of moving them to the theoretical models section, but not sure how I would convey that these constitute variabilities by doing so, except for adding text in the description of IM1. I removed the h_k parameter, as I'm not considering variable step size. —AS]

Number	IM1
Label	**One-step IVP solver**
Equations	• $\mathbf{x}_{k+1} = \mathbf{x}_k + h\phi(t_k, \mathbf{x}_k)$
Description	The above equation provides the skeleton can be used to calculate (an approximation of) a new point given the previous or starting point, given some definition of the ϕ function, two of which are given below (IM2, IM3). $\mathbf{f}(t_k, \mathbf{x}_k)$ is another variability, since each definition of $\mathbf{f}(t_k, \mathbf{x}_k)$ will yield another program family member. In addition to the ODE (DD1) this also requires an interval (DD2), step size (DD4) and initial condition (DD3).
Source	Dr. W. S. Smith
Ref. By	IM2, IM3

Number	IM2
Label	**Second-order Runge-Kutta method (Improved Euler method)**
Equations	• $t_{k+1} = t_k + h$ • $\mathbf{Y}_1 = \mathbf{x}_k + h\mathbf{f}(t_k, \mathbf{x}_k)$ • $\phi(t_k, \mathbf{x}_k) = \left(\frac{1}{2}\mathbf{f}(t_k, \mathbf{x}_k) + \frac{1}{2}\mathbf{f}(t_{k+1}, \mathbf{Y}_1)\right)$
Description	The above equations lead to the definition of a function ϕ, which can be used to calculate (an approximation of) a new point given the previous point in conjunction with IM1. In addition to the ODE (DD1) this also requires an interval (DD2), step size (DD4) and initial condition (DD3).
Source	Corless and Fillion (2013) p. 616
Ref. By	A2

Number	IM3
Label	**Fourth order Runge-Kutta method (RK4)**
Equation	• $t_{k+1} = t_k + h$ • $\mathbf{k}_1 = \mathbf{f}(t_k, \mathbf{x}_k)$ • $\mathbf{k}_2 = \mathbf{f}(t_k + \frac{h}{2}, \mathbf{x}_k + \frac{h}{2}\mathbf{k}_1)$ • $\mathbf{k}_3 = \mathbf{f}(t_k + \frac{h}{2}, \mathbf{x}_k + \frac{h}{2}\mathbf{k}_2)$ • $\mathbf{k}_4 = \mathbf{f}(t_k + h, \mathbf{x}_k + h\mathbf{k}_3)$ • $\phi(t_k, \mathbf{x}_k) = \frac{1}{6}(\mathbf{k}_1+2\mathbf{k}_2+2\mathbf{k}_3+\mathbf{k}_4)$ new point created when extrapolating from point $\mathbf{x}_k$ using a weighted average of the previously calculated slopes
Description	The above equations lead to the definition of a function ϕ, which can be used to calculate (an approximation of) a new point given the previous point in conjunction with IM1. [Why specify the first step? Why not specify the kth step? —SS] [I copied a textbook example and forgot to generalize it, I've updated it to reflect the kth step. —AS] This method starts by calculating the slope at x_k, represented by $\mathbf{k}_1$. Extrapolating this slope $\mathbf{k}_1$ from point $\mathbf{x}_k$, it calculates a slope for the point halfway between t_k and t_{k+1}. Subsequently, it calculates a better slope for the point halfway between t_k and t_{k+1} when extrapolating slope $\mathbf{k}_2$ from point $\mathbf{x}_k$. Finally, it calculates the slope of the point at t_{k+1} by extrapolating slope $\mathbf{k}_3$ from point $\mathbf{x}_k$. In addition to the ODE (DD1), this also requires an interval (DD2), step size (DD4) and initial condition (DD3).
Source	Corless and Fillion (2013) p. 618
Ref. By	A2

[Again, I'm not sure how splines fit into this? —SS] [Mostly via my optimism at the time, so I've removed the reference to it. —AS]

[Why isn't the function f listed as a variability? You get a different program depending on the value of f. —SS] [I've added this as part of the description of 1. —AS]

6.3 Output

Every program family member consists of a function that will output a numerical approximation of an ODE at a particular point on the given interval (which should be a multiple of the step size added to the lower bound of the interval).

The accuracy of the generated function may vary depending the family member that produced it and the ODE that was approximated.

7 Traceability Matrices and Graphs

The traceability matrices below show the relationships between the various concepts defined in this document. Rows may be affected by changes in the items the columns consist of (especially in the context of assumptions.)

	DD1	DD2	DD3	DD4	T1	IM1	IM2	IM3
DD1					✓			
DD2					✓			
DD3	✓				✓			
DD4		✓			✓			
T1								
IM1	✓	✓	✓	✓	✓			
IM2	✓	✓	✓	✓	✓	✓		
IM3	✓	✓	✓	✓	✓	✓		

Table 1: Traceability matrix between instance models, data definitions and theory

	A1	A2	A3	A4
DD1		✓	✓	
DD2		✓		✓
DD3			✓	
DD4				
T1				
IM1				
IM2	✓	✓		
IM3	✓	✓		

Table 2: Traceability matrix for assumptions

References

Robert M. Corless and Nicolas Fillion. *A Graduate Introduction to Numerical Methods*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-8452-3 978-1-4614-8453-0. URL http://link.springer.com/10.1007/978-1-4614-8453-0. DOI: 10.1007/978-1-4614-8453-0.

Lei Lai. Requirements documentation for engineering mechanics software: Guidelines, template and a case study. Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2004.

W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006. URL http://www.ifi.unizh.ch/req/events/RE06/.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Àgerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.