Module Interface Specification for SpecSearch

Robert E. White

November 14, 2018

1 Revision History

Date	Version	Notes
2018-11-09	1.0	Creation of first draft for presentation.

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for SpecSearch. SpecSearch is scientific computing software that computes and plots the spectrum of a matrix operator that appears in a LAX pair that is compatible for solutions of the Non-Linear Schr. . . odinger (NLS) equation. Consequently, this spectrum carries useful information regarding the stability of solutions to the NLS equation. Physicists are interested in this spectrum as the NLS equation models many physical phenomena, such as rogue waves or modulated wave packets. Mathematicans are interested in the analytical behaviour of this spectrum.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at . . . . [provide the url for your repo —SS]

The purpose of this document is to describe the intended behaviour of the access routines for SpecSearch's modules. In particular, mathematical notation will be used to describe the input/output relationships and external behaviour of the modules. The MIS is still abstract since it does not cover implementation of the modules and does not outline any code.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Program Name.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
complex	$\mathbb{C}$	any number $x+iy$ with $x \in \mathbb{R}$, $y \in \mathbb{R}$ and $i^2 = -1$.

The specification of Program Name uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Program

Name uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Output Format Spectrum Matrix Exact Eigenvalue Equations Spectrum Error Equation Numerical Parameters Control
Software Decision	Sequence Data Structure Eigenvalue and Eigenvector Solver Diagonal Matrix Generator Elliptic Integral Elliptic Functions Plotting Linspace

Table 1: Module Hierarchy

6 MIS of Input Parameters

The secrets of this module are the data structure for input parameters and methods for verifying input.

6.1 Module

InParams

6.2 Uses

-

6.3 Syntax

Name	In	Out	Exceptions
Call_params	$(k, N) \in \mathbb{R}$ x $\mathbb{N}$	-	NonNumericalError
Verify_params	-	-	BadkRange, BadNRange
k	-	$\mathbb{R}$	-
N	-	$\mathbb{N}$	-

6.4 Semantics

6.4.1 State Variables

$k \in \mathbb{R}$
$N \in \mathbb{N}$

6.4.2 Environment Variables

InputParameters: The two values entered by the user.

6.4.3 Assumptions

- Call_params is called before the values of any state variables will be accessed.
- The user inputs the state variables in the following order: (k, N)

6.4.4 Access Routine Semantics

InParams.k():

- output: $out = k$
- exception: None

InParams.N():

- output: $out = N$
- exception: None

Call_Params():

- transition: The data is read sequentially from the command line. The data are seperated by the return key. This data is used to populate the previously mentioned state variables.
- exception: If non-numerical data is entered.

Verify_Params():

- exception: exc:=

$\neg(k \in (0,1)) \Rightarrow$ BadkRange
$\neg(N \in \mathbb{N}) \Rightarrow$ BadNRange

6.4.5 Local Functions

[cells and data structures in MATLAB?? —RW]

7 MIS of Output Format

The secret of this module is the format and structure of the output data.

7.1 Module

OutForm

7.2 Uses

Eigenvalue solver, Spectrum Error Equations and Numerical Parameters.

Name	In	Out	Exceptions
capture	$(D, Er, V, Vl) \in \mathbb{R}^{2n}\text{x}\mathbb{R}\text{x}M_{4n(4n+1)}(\mathbb{C})\ \text{x}\mathbb{C}^{4n}$	$Spectrum$	-
Spectral	-	$M_{4n(4n+2)}(\mathbb{C})$	-
D	-	$\mathbb{R}^{2n}$	-
Er	-	$\mathbb{R}$	-

7.3 Semantics

7.3.1 State Variables

Spectral $\in M_{4n(4n+1)}(\mathbb{C})$
$D \in \mathbb{R}^{2n}$
$Er \in \mathbb{R}^2$

7.3.2 Assumptions

- Eigenvalue solver, Spectrum Error Equations and Numerical Parameters are called and their values are temporarily stored prior to OutForm.

7.3.3 Access Routine Semantics

OutForm.D():

- output: $out = D$
- exception: None

OutForm.Er():

- output: $out = Er$

- exception: None

OutForm.Spectral():

- output: $out = Spectral$
- exception: None

capture():

- Transition: The data (D,Er,V,Vl) will be captured from the other modules and then stored into Spectral for convienience. Spectral is a $4n$ by $4n + 2$ matrix created as follows: the $4n$ by 1 vector of eigenvalues, Vl, will be transposed and turned into the $(4n + 1)^{th}$ row of spectral. The previous $4n$ rows of spectral will be the matrix, V, composed of the eigenvectors as columns. D will be the $(4n + 2)$ row of spectral.
- Exception: None

7.3.4 Local Functions

[what are local functions?? —RW]

8 MIS of Spectrum Matrix

The structure of the spectrum matrix, its data entries, how it is created, and the numerical method for approximating its eigenfunctions are the secrets of this module.

8.1 Module

SpecMat

8.2 Uses

This module uses Numerical Parameters.

8.3 Syntax

Name	**In**	**Out**	**Exceptions**
create	$(n, h, elipdn, k) \in \mathbb{N}\text{x}\mathbb{R}\text{x}\mathbb{R}^{2n}\text{x}\mathbb{R}$	SpecMatO1, SpecMatO4, SpecCheb	-
SpecMatO1	-	$M_{4n(4n)}(\mathbb{C})$	-
SpecMatO4	-	$M_{4n(4n)}(\mathbb{C})$	-
SpecCheb	-	$M_{4n(4n)}(\mathbb{C})$	-

8.4 Semantics

8.4.1 State Variables

SpecMatO1, SpecMatO4, SpecCheb $\in M_{4n(4n)}(\mathbb{C})$

8.4.2 Assumptions

- All of the numerical parameters are calculated correctly and used as arguments in create.

8.4.3 Access Routine Semantics

SpecMat.SpecMatO1:

- output: out = SpecMatO1
- exception: None

SpecMat.SpecMatO4:

- output: out = SpecMatO4

- exception: None

SpecMat.SpecFour:

- output: out = SpecCheb
- exception: None

SpecMat.create():

- transition: The data (n,h,elipdn,k) will be captured from the other modules and used to create the spectral matrices. There will be three instances of the spectral matrix. Each corresponds do a different numerical algorithm for approximating the eigenfunction derivatives. Top top left and bottom right sections of the matrices are diagonal matrices with elements equal to the elipdn function evaluated at points in the discretized domain. The remain parts of the matrix are coffecients such that when multiplied by the eigenfunctions will create the appropriate approximation of the derivatives.
- exception: None

9 MIS of Exact Eigenvalue Equations

The secrets of this module are the analytical expressions for the two real eigenvalues.

9.1 Module

TheorEigenValues

9.2 Uses

This module uses input parameters.

9.3 Syntax

Name	In	Out	Exceptions
$lambda_O1$	$k \in \mathbb{R}$	$\mathbb{R}$	-
$lambda_O2$	$k \in \mathbb{R}$	$\mathbb{R}$	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

- These are the eigenvalues computed from segal et al. Refer to (segal) for more details regarding assumptions.

9.4.4 Access Routine Semantics

$lambda_O1$:

- output: $\frac{1}{2}(1 + \sqrt{1 - k^2})$
- exception: None

$lambda_O2$

- output: $\frac{1}{2}(1 - \sqrt{1 - k^2})$
- exception: None

10 MIS of Spectrum Error Equation

The secret of this module is the equation for error between exact and approximated eigenvalues.

10.1 Module

ErrCalc

10.2 Uses

This module uses Exact Eigenvalue equations and Eigenvalue solver.

10.3 Syntax

Name	In	Out	Exceptions
$lambda_O 1$	-	$\mathbb{R}$	-
$lambda_O 2$	-	$\mathbb{R}$	-
$lambda_C 1$	-	$\mathbb{R}$	-
$lambda_C 2$	-	$\mathbb{R}$	-
$Err1$	$(lambda_O 1, lambda_C 1) \in \mathbb{R}^2$	$\mathbb{R}$	-
$Err2$	$(lambda_O 2, lambda_C 2) \in \mathbb{R}^2$	$\mathbb{R}$	-

10.4 Semantics

10.4.1 State Variables

$(lambda_O 1, lambda_O 2, lambda_C 1, lambda_C 2) \in \mathbb{R}^4$

10.4.2 Environment Variables

None

10.4.3 Assumptions

- These are the eigenvalues computed from segal et al. Refer to (segal) for more details regarding assumptions.

10.4.4 Access Routine Semantics

$Err1$:

- output: $|lambda_O 1 - lambda_C 1|$

- exception: None

$Err2$

- output: $|lambda_O2 - lambda_C2|$
- exception: None

$ErrCalc.lambda_O1$:

- output: out = $lambda_O1$
- exception: None

$ErrCalc.lambda_O2$:

- output: out = $lambda_O2$
- exception: None

$ErrCalc.lambda_C1$:

- output: out = $lambda_C1$
- exception: None

$ErrCalc.lambda_C2$:

- output: out = $lambda_C2$
- exception: None

11 MIS of Numerical Parameters

The secrets of this module are the range of the eigenfunction domain, points in the periodic domain and equation for the numerical scaling factor that computes the eigenfunction derivatives.

11.1 Module

Numpars

11.2 Uses

This module uses Elliptic Functions, Diagonal Matrix, linspace, Elliptic Integral and input parameters.

11.3 Syntax

Name	In	Out	Exceptions
CreateVars	$(k, N) \in \mathbb{R}\text{x}\mathbb{N}$	$(xend, Domain, ellipj, ellipjMAT)$	-
xend	-	$\mathbb{R}$	-
Domain	-	$\mathbb{R}^{2N}$	-
ellipjdn	-	$\mathbb{R}^{2N}$	-
ellipjMAT	-	$M_{2n(2n)}(\mathbb{R})$	-
h	-	$\mathbb{R}$ -	

11.4 Semantics

11.4.1 State Variables

$(xend, Domain, ellipjdn, ellipjMAT) \in \mathbb{R}\text{x}\mathbb{R}^{2N}\text{x}\mathbb{R}^{2N}\text{x}M_{2n(2n)}(\mathbb{R})$

11.4.2 Environment Variables

None

11.4.3 Assumptions

- Input parameters is called before Numerical parameters.
- Input parameters does not throw an exception.

11.4.4 Access Routine Semantics

$Numpars.xend$:

- output: $xend$
- exception: None

$Numpars.Domain$

- output: $Domain$
- exception: None

$Numpars.ellipjdn$:

- output: out $= ellipjdn$
- exception: None

$Numpars.ellipjMAT$:

- output: out $= ellipjMAT$
- exception: None

$Numpars.h$:

- output: out $= h$
- exception: None

$Numpars.CreateVars$:

- transition: The data (N, k) will be captured from the input parameters and used to create the state variables. k will be inputted as an argument into the elliptic integral module. The resulting integral is equal to $xend$.
  The domain is created using linspace. The endpoint arguments of linspace are $-xend$ and $xend$, respectively. The distance between partition points in the resulting domain is $h = \frac{xend}{N}$. Ellipjdn is derived from from computing the ellipjdn value of each point in Domain. EllipjMAT is a diagonal matrix whose diagonal is Ellipjdn.
- exception: None

12 MIS of Eigenvalue and Eigenvector Solver

The secret of this module is the numerical algorithm for calculating the eigenvalues and eigenvectors of an n by n matrix.

12.1 Module

eig

12.2 Uses

This module uses Spectrum Matrix.

12.3 Syntax

Name	**In**	**Out**	**Exceptions**
solver	$A \in M_{(nxn)}(\mathbb{R})$	$\mathbb{C}^n \text{x} M_{(nxn)}(\mathbb{R})$	NotsquareMat

12.4 Semantics

12.4.1 State Variables

None.

12.4.2 Environment Variables

None.

12.4.3 Assumptions

- The input is a square matrix.

12.4.4 Access Routine Semantics

eig():

- output: out:= $(\lambda, \bar{v})$ such that:
  $A\bar{v} = \lambda\bar{v}$

- exception: exce:= $A \notin M_{(nxn)}(\mathbb{R}) \Rightarrow NotSquareMatrix$

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

13 Appendix

[Extra information if required —SS]