CAS 741: Test Plan

Aqueous Speciation Diagram Generator

Steven Palmer
`palmes4`

December 18, 2017

Revision History

Table 1: Revision History

Date	Developer(s)	Change
10.13.2017	S. Palmer	First revision of document
10.25.2017	S. Palmer	Revision 0 submission
12.18.2017	S. Palmer	Revision 1

Symbols, Abbreviations and Acronyms

symbol	description
SpecGen	The Aqueous Speciation Diagram Generator program
SRS	Software Requirements Specification
T	Test

Contents

1 General Information

This document provides a detailed description of the testing that will be carried out on the Aqueous Speciation Diagram Generator program (herein referred to as SpecGen). Complementary documents include the System Requirement Specifications, Module Guide and Module Interface Specification. The full documentation and implementation can be found here.

1.1 Purpose

The purpose of this document is to provide a comprehensive plan for testing the SpecGen software against the requirements described in the SpecGen SRS.

1.2 Scope

The test plan is narrowed to the following scope:

- The tests outlined in this document are limited to the verification of the requirements given in the SpecGen SRS. The validation of the requirements will be carried out via correspondence with Dr. Scott Smith (Wilfrid Laurier University).
- The tests outlined in this document are limited to dynamic tests only. Due to the small size and low complexity of the SpecGen program, no formal static testing (code walkthroughs, code inspections, etc.) will be carried out.
- The SpecGen software will be written in Python. The testing of implementations in other languages will not be considered in this document.

2 Plan

2.1 Software Description

Chemical speciation refers to the stable (equilibrium) distribution of chemical species in a given chemical system. Speciation diagrams, which plot species concentrations against an independently varied parameter of the system, are useful tools for displaying speciation data in a concise and easy to use format.

SpecGen will produce a speciation diagram given a set of chemical reactions, equilibrium constants, and element totals that define a chemical system. SpecGen will be specific to speciation of ions in aqueous systems under varying pH, which is of particular importance in the fields of aqueous process engineering and hydrometallurgy. The diagram generated by SpecGen will plot speciation of all aqueous species (excluding H^+ and OH^-) across the pH range 0 to 14.

2.2 Test Team

The test team includes the following members:

- Steven Palmer

2.3 Automated Testing Approach

The automated testing for SpecGen will be carried out using a set of unit and integration tests. A test coverage analysis of these tests will be carried out to ensure that testing is as complete as possible. The target for this analysis is 100% statement coverage.

Regression testing will be used during the implementation stage and for any future changes. Since SpecGen is small in scope and will be implemented by a single developer, other forms of automated testing, such as continuous integration testing, will not be considered.

2.4 Verification Tools

The following tools will be used to facilitate testing:

1. **PyTest** (a unit testing framework for Python) will be used to write and run unit tests
2. **Coverage.py** will be used to assess test coverage
3. **make** will be used to automate the building and execution of the test program

2.5 Non-Testing Based Verification

N/A

3 System Test Description

3.1 Tests for Functional Requirements

T1: Diagram generation of $FeOH_3$ system

Type: Functional, Automatic, Integration

Initial State:

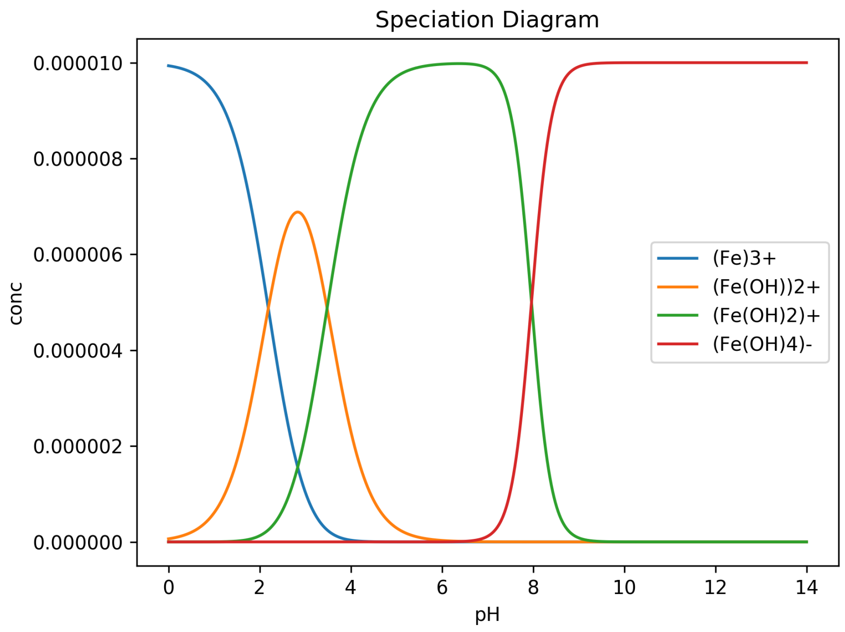
```
feSys = ChemSys()
feSys.registerRxn(
  "(Fe)3+ + (H2O)l = (Fe(OH))2+ + (H)+ , -2.19"
)
```

```
feSys.registerRxn(
  "(Fe)3+ + 2(H2O)l = (Fe(OH)2)+ + 2(H)+ , -5.67"
)
feSys.registerRxn(
  "(Fe)3+ + 4(H2O)l = (Fe(OH)4)- + 4(H)+ , -21.6"
)
feSys.registerTotal(
  "Fe", 0.000010
)
```

Input:

```
feSys.specGen("fe")
```

Output:



How test will be performed: Automated integration test

T2: Diagram generation of CO_2 system

Type: Functional, Automatic, Integration

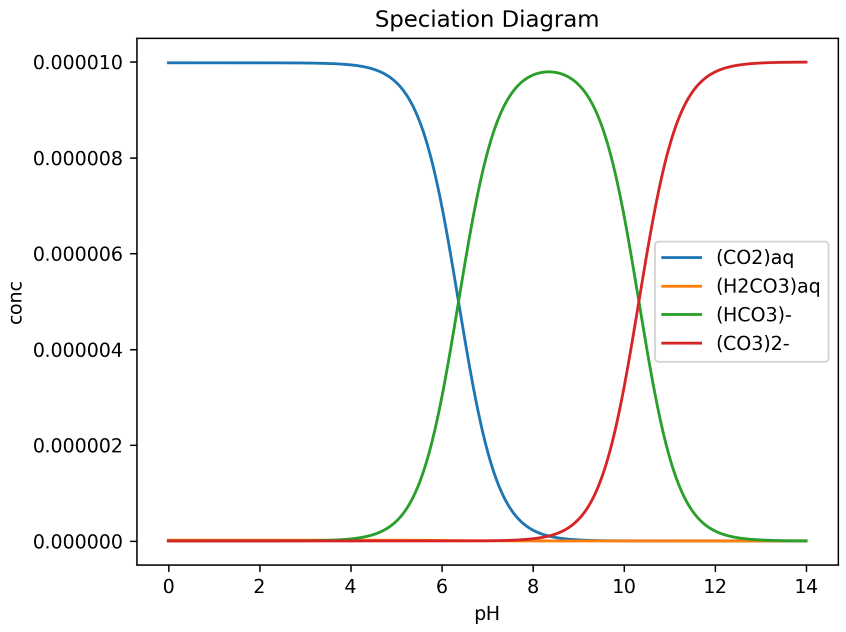
Initial State:

```
co2Sys = ChemSys()
co2Sys.registerRxn(
  "(CO2)aq + (H2O)l = (H2CO3)aq , -2.77"
)
co2Sys.registerRxn(
  "(H2CO3)aq = (HCO3)- + (H)+ , -3.6"
)
co2Sys.registerRxn(
  "(HCO3)- = (CO3)2- + (H)+ , -10.33"
)
co2Sys.registerTotal(
  "C", 0.000010
)
```

Input:

```
feSys.specGen("co2")
```

Output:



How test will be performed: Automated integration test

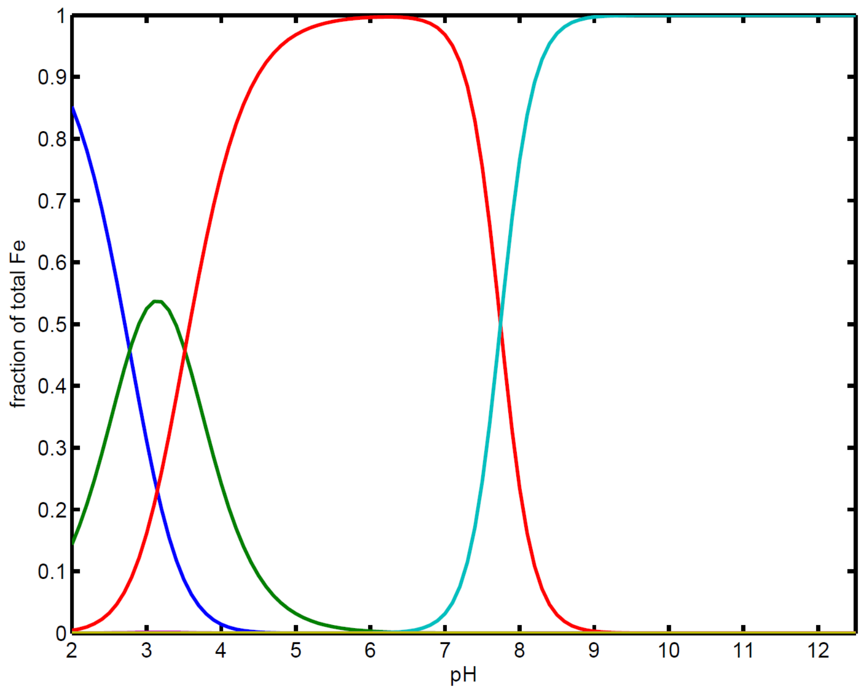
T3: Comparison of generated speciation diagram to original

Type: Functional, Manual

How test will be performed: This test will compare the diagram generated by SpecGen for the $FeOH_3$ system with the diagram generated by Dr. Smith's MATLAB implementation. The diagram will be generated in SpecGen by the following inputs:

```
feSys = ChemSys()
feSys.registerRxn(
  "(Fe)3+ + (H2O)l = (Fe(OH))2+ + (H)+ , −2.19"
)
feSys.registerRxn(
  "(Fe)3+ + 2(H2O)l = (Fe(OH)2)+ + 2(H)+ , −5.67"
)
feSys.registerRxn(
  "(Fe)3+ + 4(H2O)l = (Fe(OH)4)− + 4(H)+ , −21.6"
)
feSys.registerTotal(
  "Fe", 0.000010
)
feSys.specGen("fe")
```

The diagram should be the same as the following:



3.2 Tests for Nonfunctional Requirements

T4: Readability of generated speciation diagram

Type: Nonfunctional, Manual

How test will be performed: This is a qualitative test to ensure that the diagrams generated by SpecGen are readable (axis labels visible, curves distinguishable from each other, legend/labelling of curves, etc.).

4 Traceability Between System Tests and Requirements

A trace between system tests and requirements is provided in Table 2.

Table 2: Requirements Traceability

Requirement	**Test(s)**
R1	T1, T2, T3
R2	T1, T2, T3
R3	T1, T2, T3
R4	T1, T2, T3
R5	T1, T2, T3
NF1	T4

5 Unit Testing Plan

5.1 Plotting Module Testing

T5: Plotting Test

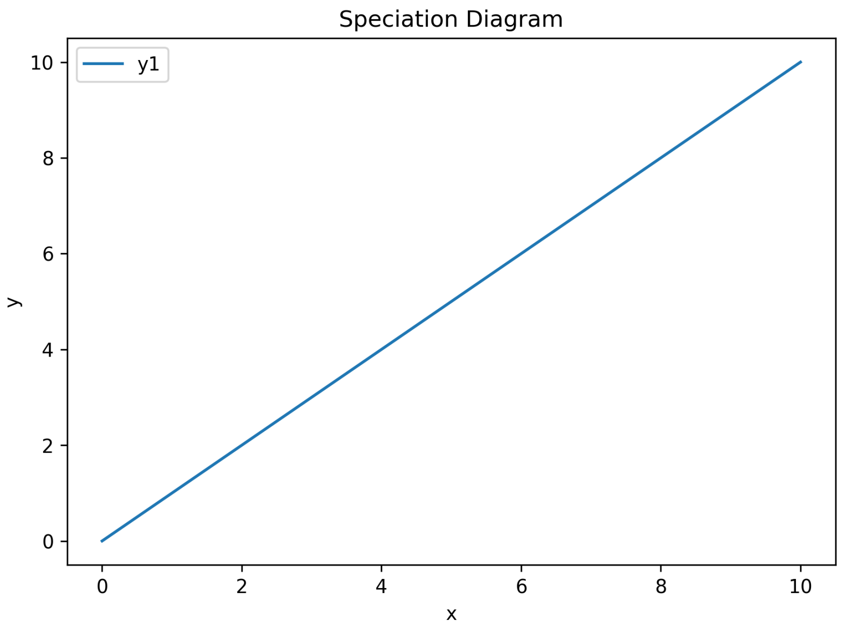
Type: Automatic, Unit

Initial State: N/A

Input:

```
genDiagram( "test_plot",
            range(0, 11),
            "x",
            [range(0, 11)],
            "y",
            ["y1"] )
```

Output:



How test will be performed: Automated unit test

5.2 Input Conversion Module Testing

T6: Input conversion of $FeOH_3$ system

Type: Automatic, Unit

Initial State:

```
feSys = ChemSys()
feSys.registerRxn(
  "(Fe)3+ + (H2O)l = (Fe(OH))2+ + (H)+ , -2.19"
)
feSys.registerRxn(
  "(Fe)3+ + 2(H2O)l = (Fe(OH)2)+ + 2(H)+ , -5.67"
)
feSys.registerRxn(
  "(Fe)3+ + 4(H2O)l = (Fe(OH)4)- + 4(H)+ , -21.6"
)
feSys.registerTotal(
  "Fe", 0.000010
)
(fn, _) = makeRootFunc(feSys)

def expectedFe(p, *arg):
  (h, oh) = arg
  (x1, x2, x3, x4) = p
  return ( x2 + h - x1 + 2.19,
           x3 + 2*h - x1 + 5.67,
           x4 + 4*h - x1 + 21.6,
           log10(10**x1 + 10**x2 + 10**x3 + 10**x4)
             - log10(0.000010) )
```

Input:

```
fn((a, b, c, d), e, f)
```

For a, b, c, d, e, f in range -14 to 0.

Output:

```
expectedFe((a, b, c, d), e, f)
```

For same values of a, b, c, d, e, f.

How test will be performed: Automated unit test

T7: Input conversion of CO_2 system

Type: Automatic, Unit

Initial State:

```
co2Sys = ChemSys()
co2Sys.registerRxn(
  "(CO2)aq + (H2O)l = (H2CO3)aq , -2.77"
)
co2Sys.registerRxn(
  "(H2CO3)aq = (HCO3)- + (H)+ , -3.6"
)
co2Sys.registerRxn(
  "(HCO3)- = (CO3)2- + (H)+ , -10.33"
)
co2Sys.registerTotal(
  "C", 0.000010
)
(fn, _) = makeRootFunc(co2Sys)

def expectedCO2(p, *arg):
  (h, oh) = arg
  (x1, x2, x3, x4) = p
  return ( x2 - x1 + 2.77,
           x3 + h - x2 + 3.6,
           x4 + h - x3 + 10.33,
           log10(10**x1 + 10**x2 + 10**x3 + 10**x4)
             - log10(0.000010) )
```

Input:

```
fn((a, b, c, d), e, f)
```

For a, b, c, d, e, f in range -14 to 0.

Output:

```
expectedCO2((a, b, c, d), e, f)
```

For same values of a, b, c, d, e, f.

How test will be performed: Automated unit test

T8: Input conversion of empty system

Type: Automatic, Unit

Initial State:

```
emptySys = ChemSys()
```

Input:

```
makeRootFunc(emptySys)
```

Output: Runtime Error

How test will be performed: Automated unit test

5.3 Calculation Module Testing

T9: Calculation of empty system

Type: Automatic, Unit

Initial State:

```
emptySys = ChemSys()
```

Input:

```
calcSpec(emptySys)
```

Output: Runtime Error

How test will be performed: Automated unit test

T10: Calculation of simple system

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
cs.registerRxn("(T)aq = (H)+ , 0")
```

Input:

```
(_, out, _) = calcSpec(cs)
out
```

Output:

```
[[10**(-x/100) for x in range(0, 1401)]]
```

How test will be performed: Automated unit test

5.4 Chemical System Module Testing

T11: Register reaction as empty string

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("")
```

Output: Value Error

How test will be performed: Automated unit test

T12: Register reaction without equilibrium constant

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("(H)+")
```

Output: Value Error

How test will be performed: Automated unit test

T13: Register reaction without products

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("(H)+  ,  0")
```

Output: Value Error

How test will be performed: Automated unit test

T14: Register reaction with bad state

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("(H)bad = (H)bad , 0")
```

Output: Runtime Error

How test will be performed: Automated unit test

T15: Register reaction with bad formula (non-letter symbol)

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("(H)bad = (H)bad , 0")
```

Output: Runtime Error

How test will be performed: Automated unit test

T16: Register reaction with bad formula (beginning with lower case)

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("(h)+ = (h)+ , 0")
```

Output: Runtime Error

How test will be performed: Automated unit test

T17: Register reaction with bad formula (no parentheses)

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("H+ = H+ , 0")
```

Output: Runtime Error

How test will be performed: Automated unit test

T18: Register reaction with bad formula (unbalanced parentheses)

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("((H)+ = (H)+ , 0")
```

Output: Runtime Error

How test will be performed: Automated unit test

T19: Register reaction with superfluous parentheses

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("((H))+ = (H)+ , 0")
```

Output: No Error

How test will be performed: Automated unit test

T20: Register reaction with high parenthesis nesting

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerRxn("((H(OH)2(H)))1 = (H)+ , 0")
```

Output: No Error

How test will be performed: Automated unit test

T21: Register negative element total

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerTotal("H", -1)
```

Output: Runtime Error

How test will be performed: Automated unit test

T22: Register zero element total

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerTotal("H", 0)
```

Output: Runtime Error

How test will be performed: Automated unit test

T23: Register positive element total

Type: Automatic, Unit

Initial State:

```
cs = ChemSys()
```

Input:

```
cs.registerTotal("H", 1)
```

Output: No Error

How test will be performed: Automated unit test

6 Traceability Between Unit Tests and Modules

A trace between unit tests and modules is provided in Table 3.

Table 3: Module Traceability

Module	**Test(s)**
M1	implemented by OS; no tests required
M2	external interface; no explicit testing; covered implicitly
M3	T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21, T22, T23
M4	data structure; no explicit testing; covered implicitly
M5	data structure; no explicit testing; covered implicitly
M6	T6, T7, T8
M7	T9, T10
M8	implemented by Python; no tests required
M9	T5