Parameter Stepping

(A self-contained series of lectures describing the purpose and implementation of parameter stepping in solving systems of nonlinear algebraic equations)

Table of Contents

1. An Introduction to Parameter Stepping

Solving systems of nonlinear algebraic equations can be difficult because doing so requires a good set of initial guesses, leading to convergence of the technique. Obtaining a good initial guesses becomes progressively more difficult as the dimensionality (number of unknowns) of the problem increases. Also, the required goodness of the initial guesses can be strongly problem dependent.

Parameter stepping is a technique employed to overcome issues associated with obtaining good initial guesses. In this document, we introduce parameter stepping and then solve two examples, one a mathematical example and the other an applied engineering example.

2. Parameter Stepping from Linear to Non-Linear Problems

We address two approaches in parameter stepping. In the first example, there is one nonlinear problem that we wish to solve. We linearize the equation then take gradually "grow" the nonlinear term until we return to the original problem of interest.

Consider the toy example, where we have one equation and one unknown.

$$f(x) = 0 \tag{1.1}$$

The equation has linear and nonlinear terms in it. The linear terms are grouped together. The nonlinear terms are also grouped together,

$$f(x) = f_{Lin}(x) + f_{Non-lin}(x) = 0 \tag{1.2}$$

There is no trouble solving the linear problem. For that we use linear algebra, which does not require an initial guess.

$$f_{Lin}(x) = 0 \tag{1.3}$$

The solution of this equation, we denote as x_{Lin}. We then add a parameter stepping factor, λ, to the problem.

$$f(x) = f_{Lin}(x) + \lambda f_{Non-lin}(x) = 0 \tag{1.4}$$

When $\lambda = 0$, we have the linear problem and we know the solution, x_{Lin}. When $\lambda = 1$, we have the original problem of interest to us.

In the Parameter Stepping procedure, we use a standard solution algorithm for solving nonlinear algebraic equations, such as the Newton-Raphson method with numerical approximations to the derivatives, to solve the nonlinear algebraic equation. We step through values of λ from 0 to 1 with step-sizes of $\delta\lambda$. The number of steps in moving from 0 to 1 is $1/\delta\lambda$. The size of the increment in $\delta\lambda$ depends on the particular problem. The goal is that $\delta\lambda$ be chosen small enough such that the converged solution from the previous value of λ provides a sufficiently good initial guess for the next value of λ such that the chosen solution method converges. A script for performing this parameter stepping is given below.

Script 1. paramstep_0to1.m

```
%
%  parameter stepping example 1
%
clear all;
close all;
format long;
% make current value of lambda available globally
global lambda
%
% set up lambda vector
%
lambda_lo = 0.0;
lambda_hi = 1.0;
dlambda = 1.0e-2;
lambda_vec = [lambda_lo:dlambda:lambda_hi];
nlambda = length(lambda_vec);
%
%  reserve memory for solution matrix
%
x_storage = zeros(nlambda,1);
err_storage = zeros(nlambda,1);
%
%  initial guess for linear problem
% (doesn't matter)
%
x0 = 1.0;
%
%  parameter step
%
for i_lambda = 1:1:nlambda
% call Newton-Raphson
    lambda = lambda_vec(i_lambda);
    [x,err] = nrnd1(x0);
% store result
  x_storage(i_lambda) = x;
  err_storage(i_lambda) = err;
% update initial guess
  x0 = x;
end
%
%  plot result & error
%
figure(1)
plot(lambda_vec,x_storage,'k-o');
xlabel('stepping parameter lambda');
ylabel('solution');
figure(2)
semilogy(lambda_vec,err_storage,'k-o');
xlabel('stepping parameter lambda');
ylabel('relative error');
```

Example 1.

Consider the function

$$f(x) = x - 3 - \log(1000x)\operatorname{erfc}\left(\frac{x}{\pi}\right)x^3$$

such that

$$f_{Lin}(x) = x - 3$$

and

$$f_{Non-lin}(x) = -\log(1000x)\operatorname{erfc}\left(\frac{x}{\pi}\right)x^3$$

This problem can be entered into the routine, `nrnd1.m`, which uses the Newton-Raphson method with numerical approximations to the derivative to solve for the root of one nonlinear (or linear of course) algebraic equation. The change to the code is shown below.

```
function f = funkeval(x)
global lambda
flin = x - 3;
fnonlin = -log(1000.0*x)*erfc(x/pi())*x^3;
f = flin + lambda*fnonlin;
```

Importantly, the identical global statement in the script is included here.

If we attempt to solve this equation in its original form (in which $\lambda = 1$) without parameter stepping, we really have no idea what a good initial guess may be. Let's guess that *x* is 1.0. If we run `nrnd1.m` from the command, we have

```
>> paramstep_0to1
icount = 1 xold = 1.000000e+00 f = -6.507976e+00 df = -1.093440e+01 xnew = 4.048165e-01  err = 1.000000e+02
icount = 2 xold = 4.048165e-01 f = -2.935861e+00 df = -1.524236e+00 xnew = -1.521304e+00  err = 1.266098e+00
icount = 3 xold = -1.521304e+00 f = 3.434527e+01 df = -8.646560e+01 xnew = -1.114396e+00  err = 3.656551e-01
Error using erfc
Input must be real and full.
```

In short, our guess wasn't good enough and the method failed to converge.

If we attempt to solve the same problem using parameter stepping, then we have

```
>> paramstep_0to1
icount = 1 xold = 1.000000e+00 f = -2.000000e+00 df = 1.000000e+00 xnew = 3.000000e+00  err = 1.000000e+02
icount = 2 xold = 3.000000e+00 f = 2.042810e-14 df = 1.000000e+00 xnew = 3.000000e+00  err = 6.809368e-15
...
icount = 1 xold = 7.147651e+00 f = -4.189546e-02 df = 5.750079e+00 xnew = 7.154937e+00  err = 1.000000e+02
icount = 2 xold = 7.154937e+00 f = -1.151980e-04 df = 5.718562e+00 xnew = 7.154957e+00  err = 2.815471e-06
icount = 3 xold = 7.154957e+00 f = -1.706963e-09 df = 5.718475e+00 xnew = 7.154957e+00  err = 4.171931e-11
```

and the following two plots are generated by the script `paramstep_0to1.m`.

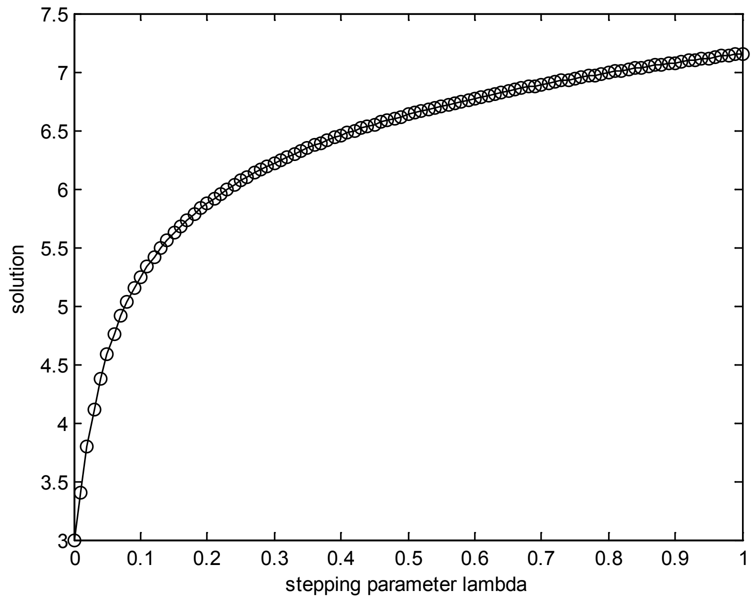

Figure 1. The solution to $f(x) = x - 3 - \lambda \log(1000x)\text{erfc}\left(\frac{x}{\pi}\right)x^3$ as a function of λ.

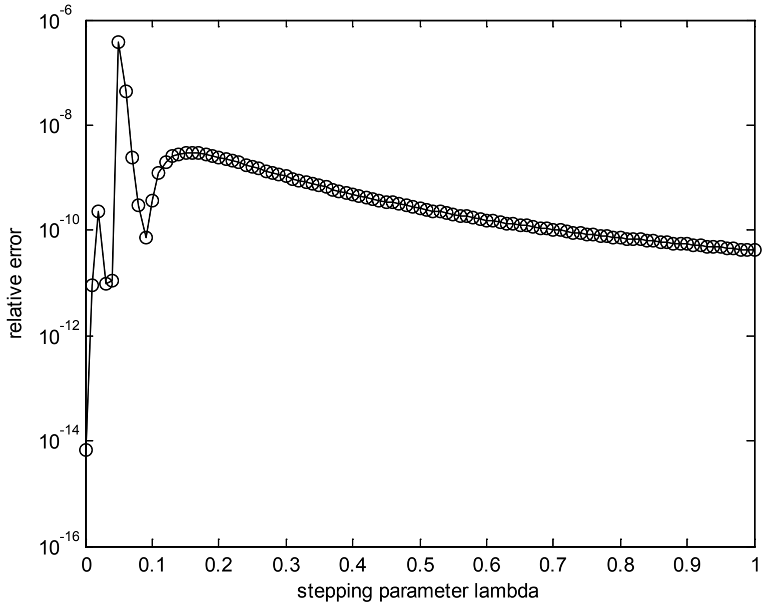

Figure 2. The error to $f(x) = x - 3 - \lambda \log(1000x)\text{erfc}\left(\frac{x}{\pi}\right)x^3$ as a function of λ.

In short, we were able to find the solution to our problem of interest (in which $\lambda = 1$), namely x = 7.15496. Also, we see that our error was always less than our tolerance of 10^{-6}, so we know that we have a converged solution, which is good to six significant digits.

3. Parameter Stepping Across a Physical Parameter

Often times, we are called upon to solve a system of nonlinear algebraic equations across a parameter space. For example, if one is generating a binary phase-diagram, one may have the following two equations, involving the chemical potential of each component, *A* and *B* in each phase, ϕ_1 and ϕ_2. The unknowns in this example are the mole fractions of component *A* in each phase, $x_A^{\phi_1}$ and $x_A^{\phi_2}$. The mole fractions of component B are not considered independent unknowns since the sum of the mole fractions within any given phase is unity, $\sum_{\alpha} x_{\alpha}^{\phi_j} = 1$ for all *j*. The thermodynamic principle of chemical equilibrium requires that the chemical potential of a component be the same in each phase.

$$f_1\left(x_A^{\phi_1}, x_A^{\phi_2}\right) = \mu_A^{\phi_1} - \mu_A^{\phi_2} = 0$$
$$f_2\left(x_A^{\phi_1}, x_A^{\phi_2}\right) = \mu_B^{\phi_1} - \mu_B^{\phi_2} = 0$$

In general the chemical potential are non-linear functions of the composition. If one is charged with building a phase diagram, one must solve this system of equations across a range of temperature. (The chemical potential are also (nonlinear) functions of temperature.)

Of course, the challenge in solving systems of nonlinear algebraic equations is coming up with good initial guesses. If we have to solve these equations for 100 different temperatures, we certainly don't want to be forced to come up with 100 sets of initial compositions good enough to allow the Newton-Raphson method to converge at each temperature. Parameter stepping through temperature provides a way to come up with a single good initial guess then to step along the temperature using the converged solution at the previous temperature as the initial guess at the next temperature.

Frequently in these examples, the numerical technique has difficulty converging at the extremes (very low temperature or very high temperature). This is because the variables, mole fractions, in one or more phases are near a boundary (zero or one for mole fractions). If the technique explores values of the mole fraction below zero or above one, the equations may likely return an error due to the presence of log(*x*) and log(1-*x*) terms. It is easiest to get an initial guess to converge at a middle temperature. In this case, two parameter stepping loops are performed. The first starts at the middle temperature, where we managed to get a solution to the equation and steps up to higher temperatures. The second loop starts at the same middle temperature and steps down to lower temperatures. When plotted together, we have the solution over the entire temperature range.

Example 2.

In the example that follows, parameter stepping is used to explore equilibria of a set of reactions in aqueous solution. Here the parameter of interest is pH rather than temperature, but the same principle applies. This example was brought to my attention by Mr. John Salasin, a PhD student in the Department of Materials Science and Engineering at the University of Tennessee, Knoxville in the winter of 2016-2017. Like any good researcher, John wanted to reproduce a result from the literature before applying the model to his own system. In this example, we reproduce the result from the following reference:

Wen-Jiung Lee and Tsang-Tse Fang, "The effect of the molar ratio of cations and citric acid on the synthesis of barium ferrite using a citrate process", *Journal of Materials Science* **30** 1995 p. 4349-4354.

Below I include the entire script written by John Salasin (and modified gently by the author of this document). The beginning of this script follows precisely the previous script. Once the variables have been solved for, there is some appended code to reproduce figures 4 and 7 from the reference.

There are many species in this problem. John has done some preliminary elimination of variables to reduce the system of equations to four equations and four unknowns. The four equations are total mass balances on citrate, iron, barium, and nitrate. The four variables correspond to the concentration of the citrate, iron, barium and nitrate ions. Simple linear relationships related the the concentrations of other species to these species, thus they could be eliminated in the same way, as in the previous example, that in a binary mixture the mole fraction of *B* can be eliminated in terms of the mole fraction of *A*.

Script 2. paramstep_BaFe.m
```
%
%  parameter stepping example 2:  Barium Ferrite
%
clear all;
close all;
format longe;
% make current value of lambda available globally
global pHi
%
% set up pH vector
%
phlo =1.0;
phhi = 7.0;
dph = 0.001;
nph = (phhi - phlo)/dph + 1;
phvec = zeros(nph,1);
phvec=[phlo:dph:phhi]';
%
%  reserve memory for solution matrix
%
solmat = zeros(nph,4);
%
%  initial guess for low pH
%
citguess=4.411200613732259e-12;
feguess=3.960000000000000e-02;
baguess=2.500000000000000e-03;
no3nguess=3.448275862068966e-01;
x0=[citguess; feguess; baguess;no3nguess];
% set the tolerance and printing switch
tol=1.0e-8;
iprint=0; % 0 is off, 1 is on
%
%  parameter step
%
for i =1:1:nph
```

```
% call Newton-Raphson
    pHi = phvec(i);
    [x,err,f] = nrndn(x0,tol,iprint);
% store result
  solmat(i,1:4) = x(1:4);
  fmat(i) = f;
% update initial guess
  x0(1:4) = x(1:4);
end

%
%  plot result
%
figure(1)
semilogy(phvec(1:nph),solmat(1:nph,1),'k-');
hold on;
semilogy(phvec(1:nph),solmat(1:nph,2),'r-');
hold on;
semilogy(phvec(1:nph),solmat(1:nph,3),'b-');
hold on;
semilogy(phvec(1:nph),solmat(1:nph,4),'g-');
hold off;
xlabel('pH');
ylabel('solution');
legend('Citrate','Fe','Ba','Nitrate')
%
%  plot error
%
figure(2)
semilogy(phvec(1:nph),fmat(1:nph),'k-');
xlabel('pH');
ylabel('average absolute error on f');

%
%  calculate other variables to Reproduce Figure 4 of reference
%
global k1 k2 k3 k4 k5 k6 k7 k8 k9 k10

solmat4 = zeros(nph,4);
for i = 1:1:nph
    H = 10.0^(-phvec(i));
    OH = 10.0^(-(14.0-phvec(i)));
    %Fe3p
    solmat4(i,1) =solmat(i,2);
    %FeOH2n
    solmat4(i,2) =solmat(i,2)*OH*k6;
    %FeHCitp
    solmat4(i,3)=solmat(i,2)*solmat(i,1)*H*(k4/k3);
    %FeCit
    solmat4(i,4)=solmat(i,2)*solmat(i,1)*k5;
end

figure(4)
plot(phvec(1:nph),solmat4(1:nph,1),'k-');
hold on;
plot(phvec(1:nph),solmat4(1:nph,2),'r-');
```

```
hold on;
plot(phvec(1:nph),solmat4(1:nph,3),'b-');
hold on;
plot(phvec(1:nph),solmat4(1:nph,4),'g-');
title('Fe Complexes')
xlabel('pH')
ylabel('Molarity')
legend('Fe','FeOH','FeHCit','FeCit')
hold off;

%  calculate other variables to Reproduce Figure 7 of reference

solmat5 = zeros(nph,5);
for i = 1:1:nph
    H = 10.0^(-phvec(i));
    OH = 10.0^(-(14.0-phvec(i)));
    %ba2p
    solmat5(i,1) =solmat(i,3);
    %bano3n
    solmat5(i,2) =solmat(i,3)*solmat(i,4)*k10;
    %bacitn
    solmat5(i,3)=solmat(i,3)*solmat(i,1)*k9;
    %bahcit
    solmat5(i,4)=solmat(i,3)*solmat(i,1)*H*(k8/k3);
    %bah2cit
    solmat5(i,5)=solmat(i,3)*solmat(i,1)*H^2*(k7/(k2*k3));
end

figure(7)
plot(phvec(1:nph),solmat5(1:nph,1),'k-');
hold on;
plot(phvec(1:nph),solmat5(1:nph,2),'r-');
hold on;
plot(phvec(1:nph),solmat5(1:nph,3),'b-');
hold on;
plot(phvec(1:nph),solmat5(1:nph,4),'g-');
hold on;
plot(phvec(1:nph),solmat5(1:nph,5),'m-');
title('Ba Complexes')
xlabel('pH')
ylabel('Molarity')
legend('Ba','BaNO','BaCit','BaHCit','BaH2Cit')
hold off;
```

This problem can be entered into the routine, `nrndn.m`, which uses the Newton-Raphson method with numerical approximations to the derivative to solve for the root of *n* nonlinear (or linear of course) algebraic equations. The input file in which the functions were entered for nrndn.m is provided below.

```
function f = funkeval(x)
% these two lines force a column vector of length n
n = max(size(x));
f = zeros(n,1);
%
%global variables for post processing
```

```
%
global pHi
global k1 k2 k3 k4 k5 k6 k7 k8 k9 k10
%Setting input variables for solver
cit3n = x(1);
fe3p = x(2);
ba2p = x(3);
no3n=x(4);
%Known Molarities & Citrate ratio control
fe_tot = 0.12;
ba_tot = 0.01;
no3n_tot=0.36;
ratio=20/13;
h3cit_tot = (fe_tot+ba_tot)*ratio;
%H and OH calculator
pH = pHi;
pOH= 14.0-pH;
H = 10.0^(-pH);
OH = 10.0^(-pOH);
%equillibrium constants
k1=1.17*10^-3;
k2=4.17*10^-5;
k3=1.82*10^-6;
k4=5.00*10^6;
k5=1.58*10^11;
k6=1.00*10^11;
k7=6.17;
k8=56.23;
k9=776.25;
k10=8.7;
%Equation Constants
a = 1.0 + H/k3 + H^2/(k2*k3) + H^3/(k1*k2*k3);
b = H*(k4/k3)+k5;
c = H^2*(k7/(k2*k3))+H*(k8/k3)+k9;
d = 1.0+k6*OH;
e = k10;
%
%Total Cation concentration Balance equations
%
% right hand side of equation
cit_rhs = h3cit_tot;
fe3p_rhs = fe_tot;
ba2p_rhs = ba_tot;
no3n_rhs= no3n_tot;
% left hand side of equation
cit_lhs  = a*cit3n+b*fe3p*cit3n+c*ba2p*cit3n;
fe3p_lhs = d*fe3p+b*fe3p*cit3n;
ba2p_lhs = ba2p+e*ba2p*no3n+c*cit3n*ba2p;
no3n_lhs = no3n+e*ba2p*no3n;
%
% Functions = RHS - LHS = 0
%
f(1) = cit_rhs - cit_lhs;
f(2) = fe3p_rhs - fe3p_lhs;
f(3) = ba2p_rhs - ba2p_lhs;
f(4) = no3n_rhs-no3n_lhs;
```

At the command line prompt, the script can be run,

```
>> paramstep_BaFe
```

This generates the following four plots.

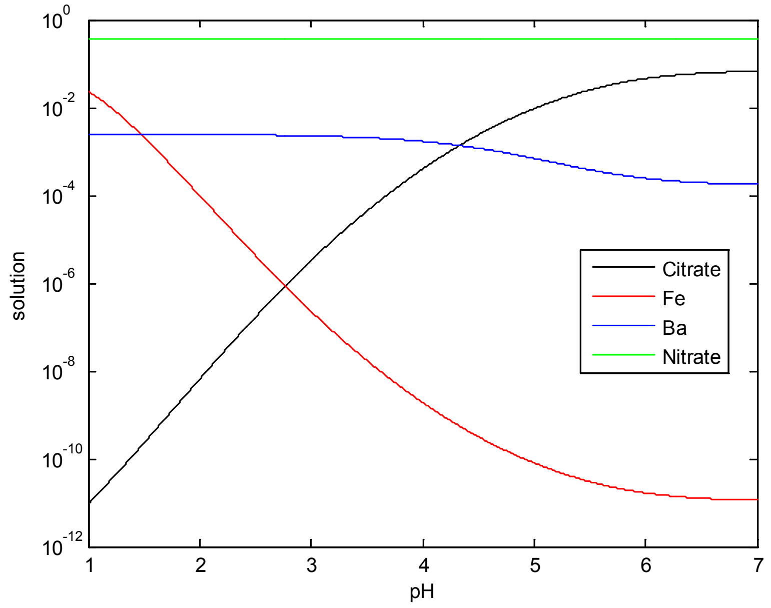

Figure 3. Solution of ion concentrations as a function of pH on a semilog plot.

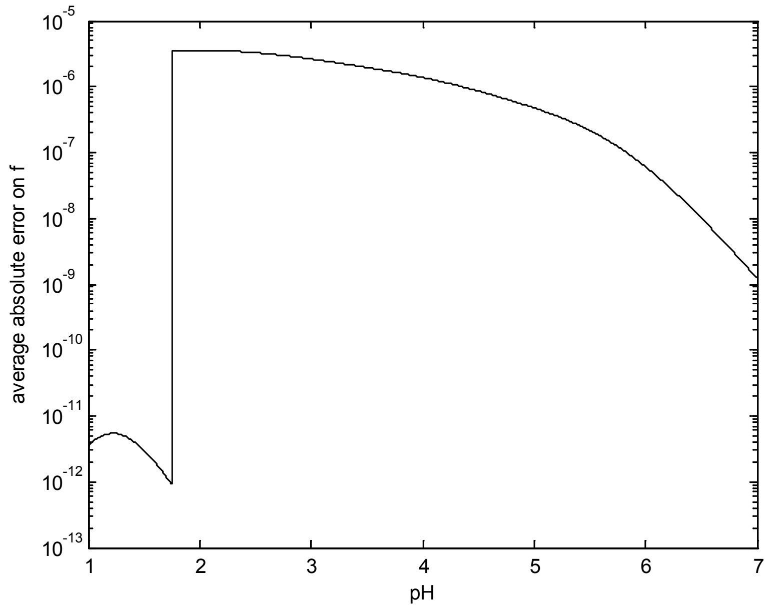

Figure 4. Plot of absolute error on the functions, *f*, vs. pH on a semilog plot. This plot clearly indicates that all points are converged.

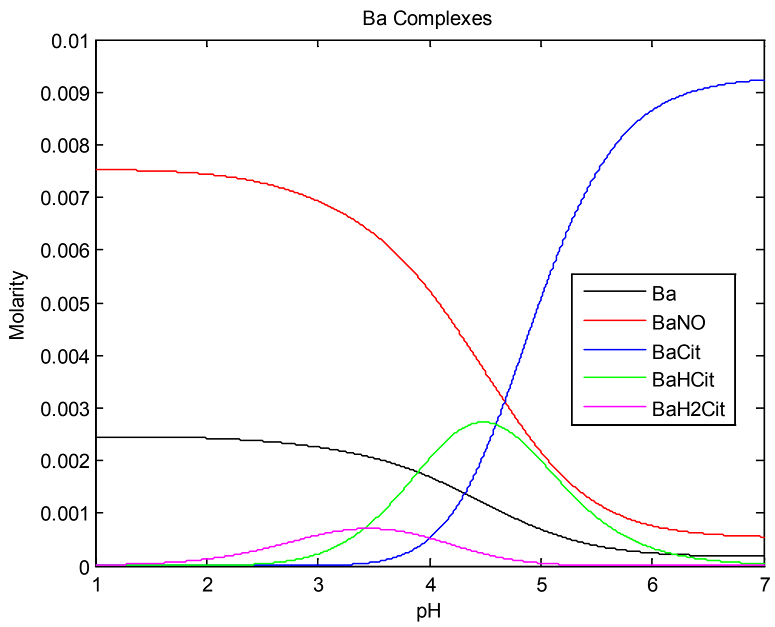

Figure 5. Concentrations of barium complexes as a function of pH. This figure reproduces Figure 4 of the reference.

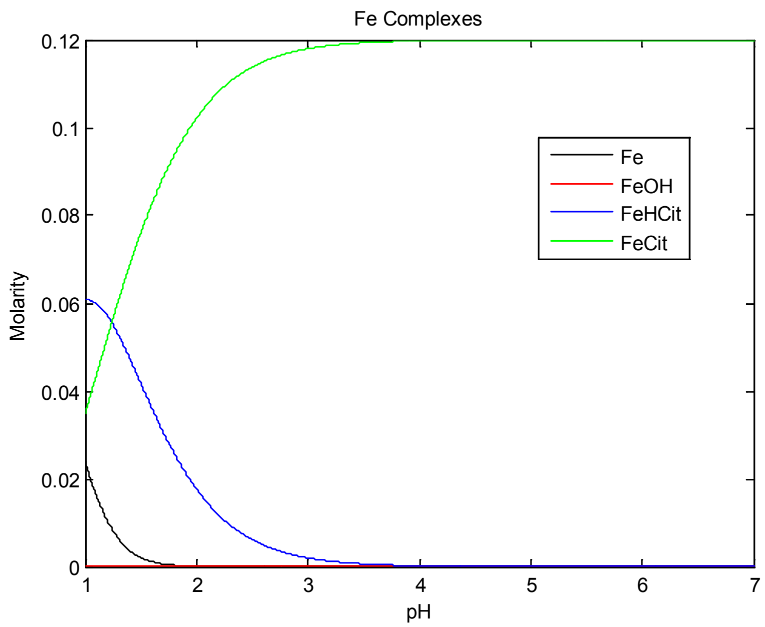

Figure 6. Concentrations of iron complexes as a function of pH. This figure reproduces Figure 7 of the reference.

Finally, we note that in this example, we used a parameter step size of 0.001 in pH. This value was chosen because larger values, such as 0.01, did not allow the process to converge across the entire pH range.