**Dynamic approach to predict pH profiles of biologically relevant buffers**

*Ganesh K1, Soumen R2, Ravichandran Y3, Janarthanan. 2,4,\**

1. Curtiss-Wright Avionics and Electronics, Dublin 14, Ireland.
2. VClinbio Labs Pvt Ltd, Sri Ramachandra Medical Centre, Porur, Chennai 600116, India
3. Department of Biotechnology, School of Bioengineering, SRM University, Kattankolathur 603203, India
4. Jimma Institute of Technology, Department of Bio-medical Engineering, Jimma University, Ethiopia.

\*Corresponding author: jana.jk2006@gmail.com

# Algebraic method for DERIVATION OF mono, poly-protic acids, phosphate and universal buffer

# mono-protic acid:

Consider the volumetric titration of a weak mono-protic acid (*HA*) titrated with a strong alkali *NaOH*. The equilibrium kinetic model can be written for the dissociation of acid (Eq. S1), alkali (Eq. S2) and water (Eq. S3) as,

|  |  |  |
| --- | --- | --- |
|  |  | (S1) |
|  |  | (S2) |
|  |  | (S3) |

Firstly, we write the equilibrium equation from Eq. S1 using Le-Chatelier’s principle,

|  |  |  |
| --- | --- | --- |
|  |  | (S4) |

From Eq. S4 we get,

|  |  |  |
| --- | --- | --- |
|  |  | (S5) |

The law of mass balance for acid is:

|  |  |  |
| --- | --- | --- |
|  |  | (S6) |

Using law of mass balance Eq. S6, we express the [*A-*] in terms of total acid [*A*]*T* by substituting Eq. S5 into Eq. S6 .

|  |  |  |
| --- | --- | --- |
|  |  | (S7) |

|  |  |  |
| --- | --- | --- |
|  |  | (S8) |

Then we express [H*A*] in terms of total acid [*A*]*T* by substituting Eq. S8 into Eq. S6.

|  |  |  |
| --- | --- | --- |
|  |  | (S9) |

Using charge neutrality we write,

|  |  |  |
| --- | --- | --- |
|  |  | (S10) |

Using Eq. S8 we rewrite Eq. S10 in terms of [*H+*] as,

|  |  |  |
| --- | --- | --- |
|  |  | (S11) |

In the above Eq. S11, the concentration of [*Na+*] was considered to be total alkali added i.e. [*NaOH*]*T*; and expression for [*OH-*] in terms of [*H+*] was obtained from the equilibrium Eq. S3, where,

|  |  |  |
| --- | --- | --- |
|  |  | (S12) |

Eq. S11 can be rewritten as a polynomial in terms of [*H+*] as,

|  |  |  |
| --- | --- | --- |
|  |  | (S13) |

By solving the above polynomial for three roots and selecting the correct one, whose value is ‘real’ (non-imaginary), ‘positive’ and its magnitude less than ; we obtain the physically relevant concentration of [*H+*] ion. The negative logarithmic value of [*H+*] to the base 10 gives the pH value (*pH=-log10* [*H+*]). By varying (which is the the cumulative concentration of alkali added till each titration point) in equation S13, we obtain different polynomials which can be solved to predict the *pH* value, so as to construct the complete titration curve.

Using symbolic tools available in matlab the polynomials can be easily derived for mono-protic acid as follows:

% Matlab code to derive Eqn 13 using symbolic tool. The variables used are, AT: Total acid ( ionized + unionized), NaOH: Total alkali, Na : Sodium ion, H :proton, OH : Hydroxyl ion, HA: Weak acid (non-dissociated/unionized), A: Weak acid (dissociated/ionized), K1: equilibrium constant for dissociation of HA, KW equilibrium constant for water dissociation.

clc;

clear;

syms AT NaOH Na H OH HA A K1 KW % declare all symbolic variables to be used

[fA,fHA]=solve(H\*A/HA == K1, A+HA == AT,A,HA) % Express A,HA in terms of H

W=solve(H\*OH==KW,OH) % Express OH in terms of H

X=simplifyFraction(H+ NaOH -W-fA,'Expand', true) % solve charge neutrality equation in H

[Y\_1,Y\_2]=numden(X) % Get the numerator which is the desired polynomial in H

Z=collect(Y\_1,H) % Express the polynomial in the order of decreasing power of H

The numerical simulation of mono-protic acid can be performed using the following code:

% Major part of this code was obtained from the webpage: http://www.chemicalforums.com/index.php?topic=80995.0

clear;

clc;

Max\_species=1;

Max\_roots=Max\_species+2;

% Equilibrium constants

KW = power(10,-14);

K1 = power(10,-4); % power(10,pKa);

% Initial concentration of acid

C\_acid = 1; % in mol/dm^3 or Molarity

C\_base = 1; % in mol/dm^3 or Molarity

% Initial volume of acid

Acid\_vol = 10; % ml

Base\_vol = 25; % ml

% Graph settings

Titration\_resolution=500; % number of titration points

dx = Base\_vol/(Titration\_resolution-1);

x = 0.0:dx:Base\_vol;

% Array for all the species

HA=size(Titration\_resolution,1);

A=size(Titration\_resolution,1);

H=size(Titration\_resolution,1);

PH=size(Titration\_resolution,1);

OH=size(Titration\_resolution,1);

FNa=size(Titration\_resolution,1);

FHA=size(Titration\_resolution,1);

Beta=size(Titration\_resolution-1,1);

outfile=fopen('Results','w');

for j = 1:Max\_roots

fprintf(outfile,'Roots(%d) ',j);

end

fprintf(outfile,'Acid\_vol Base\_vol Vol\_tot HA(i,1) A(i,1) H(i,1) OH(i,1) PH(i,1) FHA(i,1) FNa(i,1) Beta(i,1)\n');

for i = 1:Titration\_resolution

% The total volume

vol\_tot = Acid\_vol + x(i);

% Concentrations of acid and base

AT = (C\_acid \* Acid\_vol) / vol\_tot;

NaOH = (C\_base \* x(i)) / vol\_tot ;

% Compute the concentration of hydrogen ions

% Solve the polynomial equation

% H^3 + (NaOH + K1)\*H^2 + (NaOH\*K1 - AT\*K1 - KW)\*H - K1\*KW

R = roots([1, (NaOH + K1), (NaOH\*K1 - AT\*K1 - KW), - K1\*KW]);

% Remove all chemically irrelevant solutions

check\_index=1;

Temp\_H=0;

for j = 1:Max\_roots

fprintf(outfile,'%.20f ',R(j,1));

if R(j,1)==real(R(j,1)) & R(j,1)>0 & R(j,1)<AT

if check\_index==1

Temp\_H=R(j,1);

check\_index=2;

end

if Temp\_H >R(j,1)

Temp\_H=R(j,1);

end

end

end

H(i,1)=Temp\_H;

% Calculate the pH value

PH(i,1) = -log10(H(i,1));

% Calculate all other species

HA(i,1) = (AT\*H(i,1))/(H(i,1) + K1);

A(i,1) = (AT\*K1)/(H(i,1) + K1);

OH(i,1) = KW/H(i);

FHA(i,1)=AT;

FNa(i,1)=NaOH;

if(i>1)

Beta(i,1)=(FNa(i,1)-FNa(i-1,1))/(PH(i,1)-PH(i-1,1));

else

Beta(i,1)=0;

end

fprintf(outfile,'%.12f %.12f %.12f %.12f %.12f %.20f %.20f %.12f %.12f %.12f %.12f\n',Acid\_vol, x(i), vol\_tot, HA(i,1),A(i,1),H(i,1), OH(i,1), PH(i,1),FHA(i,1),FNa(i,1),Beta(i,1));

end

fclose(outfile);

% Plot it

figure();

plot(x,PH,'red')

xlabel('Volume Alkali (ml)')

ylabel('pH')

title('Titration (Volume of alkali vs pH)')

# mono-Alkaline compound:

The volumetric titration of a weak mono-alkaline compound (*BOH*) titrated with a strong acid *HCl* follows the same mathematical treatment as explained for mono-protic acid. Here, we just present the equilibrium kinetic model (Eq.S14, S15, S16), and the matlab code to simulate the mechanism as follows,

|  |  |  |
| --- | --- | --- |
|  |  | (S14) |
|  |  | (S15) |
|  |  | (S16) |

The numerical simulation of mono-alkaline compound can be performed using the following code:

% Major part of this code was obtained from the webpage: http://www.chemicalforums.com/index.php?topic=80995.0

tic

clear

clc

% Equilibrium constants

Kw = power(10,-14);

K1 = power(10,-4.6); % pka=14-pKb, 14-9.245=4.6

% Initial concentration of acid

C\_acid = 1; % mol/dm^3

C\_base = 1; % mol/dm^3

% Initial volume of acid

Acid\_vol = 25; % ml

Base\_vol = 10; % ml

% Graph settings

Titration\_resolution=500; % number of titration points

dx = Acid\_vol/(Titration\_resolution-1);

x = 0.0:dx:Acid\_vol;

% Array for all the species

BOH=size(Titration\_resolution,1);

B=size(Titration\_resolution,1);

H=size(Titration\_resolution,1);

OH=size(Titration\_resolution,1);

PH=size(Titration\_resolution,1);

FBOH=size(Titration\_resolution,1);

FHCl=size(Titration\_resolution,1);

R =size(3,1);

% Volume and Beta

BOH\_vol=size(Titration\_resolution,1);

HCl\_vol=size(Titration\_resolution,1);

Tot\_vol=size(Titration\_resolution,1);

Beta=size(Titration\_resolution,1);

outfile=fopen('Results\_acid','w');

fprintf(outfile,'Roots(3) Base\_vol Acid\_vol BOH(i,1) B(i,1) H(i,1) OH(i,1) PH(i,1) FHCl(i,1) FBOH(i,1)\n');

for i = 1:Titration\_resolution

% The total volume

vol\_tot = Base\_vol + x(i);

% Concentrations of acid and base

C\_a0 = (C\_acid \* x(i)) / vol\_tot;

C\_b0 = (C\_base \* Base\_vol) / vol\_tot ;

BOH\_vol(i,1)=Base\_vol;

HCl\_vol(i,1)=x(i);

Tot\_vol(i,1)=vol\_tot;

% Compute the concentration of hydrogen ions

% Solve the polynomial equation

% K1\*H^3 + (Kw - C\_a0\*K1 + C\_b0\*K1)\*H^2 + (- C\_a0\*Kw - K1\*Kw)\*H - Kw^2 =0

R = roots([K1, (Kw - C\_a0\*K1 + C\_b0\*K1), (- C\_a0\*Kw - K1\*Kw), -Kw^2]);

% Remove all chemically irrelevant solutions

check\_index=1;

Temp\_H=0;

for j = 1:3

fprintf(outfile,'%.20f ',R(j,1));

if R(j,1)==real(R(j,1)) && R(j,1) >=0

if check\_index==1

Temp\_H=R(j,1);

check\_index=2;

end

if Temp\_H >R(j,1)

Temp\_H=R(j,1);

end

end

end

H(i,1)=Temp\_H;

% Calculate all other species

BOH(i,1) =(C\_b0\*Kw)/(Kw + H(i,1)\*K1);

B(i,1) = (C\_b0\*H(i,1)\*K1)/(Kw + H(i,1)\*K1);

HCl(i,1)=C\_a0;

OH(i,1) = Kw/H(i,1);

FHCl(i,1)=C\_a0;

FBOH(i,1)=C\_b0;

% Calculate the pH value

PH(i,1) = -log10(H(i,1));

if(i>1)

Beta(i,1)=(FHCl(i,1)-FHCl(i-1,1))/(PH(i,1)-PH(i-1,1));

else

Beta(i,1)=0;

end

fprintf(outfile,'%.12f %.12f %.12f %.12f %.12f %.12f %.20f %.20f %.12f %.12f %.12f\n',BOH\_vol(i,1),HCl\_vol(i,1), Tot\_vol(i,1), BOH(i,1),B(i,1),H(i,1), OH(i,1), PH(i,1),FHCl(i,1),FBOH(i,1),Beta(i,1));

end

fclose(outfile);

% Plot it

figure();

plot(x,PH,'red')

xlabel('Volume Acid (ml)')

ylabel('pH')

title('Titration (Volume of acid vs pH)')

toc

# poly-protic acid (Di-Protic)

Instead of deriving, we provide the symbolic matlab code to derive the analytical expression for di-protic acid as follows,

1. % Matlab code to derive analytical expression for di-protic acid using symbolic tool. The variables used are, AT: Total acid (ionized + unionized), NaOH: Total alkali, Na : Sodium ion, H :proton, OH : Hydroxyl ion, H2A: Weak di-protic acid (non-dissociated/unionized), HA: Weak di-protic acid with one proton dissociated, A: Weak di-protic acid with two protons dissociated, K1: equilibrium constant for dissociation of H2A, K2: equilibrium constant for dissociation of HA, KW equilibrium constant for water dissociation.

clc;clear;syms AT NaOH Na H OH H2A HA A K1 K2 KW

[fA,fHA,fH2A] = solve((HA\*H)/H2A == K1, (A\*H)/HA == K2, A+HA+H2A == AT,A,HA,H2A)W = solve (H\*OH==KW,OH)X = simplifyFraction (H+NaOH-W-2\*fA-fHA,'Expand', true)[Y\_1,Y\_2] = numden (X)Z = collect(Y\_1,H)

# phosphate buffer

We provide the symbolic matlab code to derive the analytical expression for phosphate buffer as follows,

% Matlab code to derive analytical expression for phosphate buffer using symbolic tool. The variables used are

% AT : Total acid which includes unionized and ionized anionic contribution from acid and salts ( ionized + unionized), NaH2PO4 : Mono sodium phosphate, Na2HPO4 : Di sodium phosphate, Na : Sodium ion, H :proton, OH : Hydroxyl ion, H3A : H3PO4 or Phosphoric acid (non-dissociated/unionized), H2A : H2PO4- , HA : HPO42-, A: PO43-; K1,K2,K3 are the equilibrium constant for H3A, H2A, HA; KW equilibrium constant for water dissociation.

clc;clear;syms AT NaH2PO4 Na2HPO4 Na H OH H3A H2A HA A K1 K2 K3 KW

[fA,fHA,fH2A, fH3A]=solve((H2A\*H)/H3A == K1, (HA\*H)/H2A == K2, (A\*H)/HA == K3, A+HA+H2A+H3A == AT,A,HA,H2A,H3A)W=solve(H\*OH==KW,OH)X=simplifyFraction(H+NaH2PO4+ 2\*Na2HPO4-W-3\*fA-2\*fHA-fH2A,'Expand', true)[Y\_1,Y\_2]=numden(X)Z=collect(Y\_1,H)

# universal buffer

We provide the symbolic matlab code to derive the analytical expression for universal buffer as follows,

% Matlab code to derive analytical expression for universal buffer using symbolic tool. The variables used are

% H3PO4: Total phosphoric acid which includes unionized and ionized anionic contribution from acid and salts ( ionized + unionized), C6H807: Total citric acid ( ionized + unionized), H3BO3: Total boric acid ( ionized + unionized), Na3PO4: Tri sodium phosphate, Na : Sodium ion, H :proton, OH : Hydroxyl ion, H3A : H3PO4 or Phosphoric acid (non-dissociated/unionized), H2A : H2PO4- , HA : HPO42-, A: PO43-; H3B : C6H807 or Citric acid (non-dissociated/unionized), H2B : C6H707 - , HB : C6H607 2-, B: C6H507 3-; H3C : H3BO3 or Boric acid (non-dissociated/unionized), H2C : H2BO3 - , HC : HBO3 2-, C: BO3 3-; K1,K2,K3 are the equilibrium constant for H3A, H2A, HA; K4,K5,K6 are the equilibrium constant for H3B, H2B, HB; K7,K8,K9 are the equilibrium constant for H3C, H2C, HC; KW equilibrium constant for water dissociation.

clc;clear;syms H3PO4 C6H807 H3BO3 Na3PO4 Na H OH KW H3A H2A HA A H3B H2B HB B H3C H2C HC C K1 K2 K3 K4 K5 K6 K7 K8 K9[fA,fHA,fH2A,fH3A]=solve((H2A\*H)/H3A == K1, (HA\*H)/H2A == K2, (A\*H)/HA == K3, A+HA+H2A+H3A == H3PO4,A,HA,H2A,H3A) % mole-fractions of ionized species of Phosphoric acid

[fB,fHB,fH2B,fH3B]=solve((H2B\*H)/H3B == K4, (HB\*H)/H2B == K5, (B\*H)/HB == K6, B+HB+H2B+H3B == C6H807,B,HB,H2B,H3B) % mole-fractions of ionized species of Citric acid [fC,fHC,fH2C,fH3C]=solve((H2C\*H)/H3C == K7, (HC\*H)/H2C == K8, (C\*H)/HC == K9, C+HC+H2C+H3C == H3BO3,C,HC,H2C,H3C) % mole-fractions of ionized species of Boric acid

W=solve(H\*OH==KW,OH)X=simplifyFraction(H+3\* Na3PO4-W-3\*fA-2\*fHA-fH2A-3\*fB-2\*fHB-fH2B-3\*fC-2\*fHC-fH2C,'Expand', true)[Y\_1,Y\_2]=numden(X)Z=collect(Y\_1,H)

# Dynamic approach for mono-protic acid

We provide here the explicit matlab code to simulate the pH profile of mono-protic acid using differential approach.

# Simulation with IMPLICIT dissociation of Sodium hydroxide

Here the dissociation of NaOH is considered implicitly. The moment NaOH is added to the titrand, it completely dissociates to Na+ and OH-. Hence, while modelling, the concentration of NaOH is added to the initial concentration of [OH-], thereby, avoiding inclusion of an explicit rate equation for NaOH, which is an irreversible dissociation sort of reaction. The system of ordinary differential equations for 4 species such as [HA], [A-], [H+] and [OH-] can then be solved with its respective initial condition.

% Monoproticacid.m

%Takes 300sec to complete on 8 cores, 8 GB memory Dell machine

clear

clc

tic

Rate\_const\_no=4;

Diffeq\_no=4;

%Numerical integration

Custom\_RelTol=1e-10;

Custom\_AbsTol=1e-18;

Max\_time=1e18; %sec

% Equilibrium constants

Kw = power(10,-14);

K1 = power(10,-4);

% Initial concentration of acid

C\_acid = 1; % mol/dm^3

C\_base = 1; % mol/dm^3

% Initial volume of acid

Acid\_vol = 10; % ml

Base\_vol = 25; % ml

% Graph settings

Titration\_resolution=500; % number of titration points

dx = Base\_vol/(Titration\_resolution-1);

x = 0.0:dx:Base\_vol;

% Array for all the species

HA=size(Titration\_resolution,1);

A=size(Titration\_resolution,1);

H=size(Titration\_resolution,1);

PH=size(Titration\_resolution,1);

OH=size(Titration\_resolution,1);

FNa=size(Titration\_resolution,1);

FHA=size(Titration\_resolution,1);

% Volume and Beta

HA\_vol=size(Titration\_resolution,1);

NAOH\_vol=size(Titration\_resolution,1);

Tot\_vol=size(Titration\_resolution,1);

Beta=size(Titration\_resolution,1);

K=size(Rate\_const\_no);

Y\_init=size(Diffeq\_no);

Y\_tol=size(Diffeq\_no);

for j = 1:Diffeq\_no

Y\_tol(j)=Custom\_AbsTol;

end

outfile=fopen('Results','w');

fprintf(outfile,'Acid\_vol Base\_vol Tot\_vol HA(i,1) A(i,1) H(i,1) OH(i,1) PH(i,1) FHA(i,1) FNa(i,1) Beta(i,1)\n');

for i = 1:Titration\_resolution

% The total volume

vol\_tot = Acid\_vol + x(i);

% Concentrations of acid and base

C\_a0 = (C\_acid \* Acid\_vol) / vol\_tot;

C\_b0 = (C\_base \* x(i)) / vol\_tot ;

HA\_vol(i,1)=Acid\_vol;

NAOH\_vol(i,1)=x(i);

Tot\_vol(i,1)=vol\_tot;

% Compute the concentration of hydrogen ions

Y\_init(1)=C\_a0; % HA

Y\_init(2)=0; % A-

Y\_init(3)=power(10,-7); % H+

Y\_init(4)=power(10,-7)+ C\_b0; % OH-

options = odeset('RelTol',Custom\_RelTol,'AbsTol',Y\_tol);

K(1)=K1; %power(10,pKa1);

K(2)=power(10,0);

K(3)=power(10,2); % k2; dummy and not used

K(4)=Kw;

K(5)=power(10,0);

[T,Y] = ode15s(@Monoprotic\_ODE,[0,Max\_time],Y\_init,options,K);

S=size(Y);

% Calculate all other species

HA(i,1) = Y(S(1),1);

A(i,1) = Y(S(1),2);

H(i,1) = Y(S(1),3);

OH(i,1) = Y(S(1),4);

FHA(i,1)=C\_a0;

FNa(i,1)=C\_b0;

% Calculate the pH value

PH(i,1) = -log10(H(i,1));

if(i>1)

Beta(i,1)=(FNa(i,1)-FNa(i-1,1))/(PH(i,1)-PH(i-1,1));

else

Beta(i,1)=0;

end

fprintf(outfile,'%.12f %.12f %.12f %.12f %.12f %.12f %.20f %.20f %.12f %.12f %.12f\n',HA\_vol(i,1),x(i), Tot\_vol(i,1), HA(i,1),A(i,1),H(i,1), OH(i,1), PH(i,1),FHA(i,1),FNa(i,1),Beta(i,1));

i

end

fclose(outfile);

% Plot it

figure();

plot(x,PH,'red')

xlabel('Volume Alkali (ml)')

ylabel('pH')

title('Titration (Volume of alkali vs pH)')

toc

This program calls the function “Monoprotic\_ODE.m”, which is defined as,

function dy = Monoprotic\_ODE(t,y,K)

dy = zeros(5,1); % a column vector

dy(1) = -K(1)\*y(1) +K(2)\*y(2)\*y(4);

dy(2) = +K(1)\*y(1) -K(2)\*y(2)\*y(4);

dy(3) = K(1)\*y(1) -K(2)\*y(2)\*y(4)+K(4)-K(5)\*y(4)\*y(5);

dy(4) = K(4)-K(5)\*y(4)\*y(5);

# Simulation with explicit dissociation of Sodium hydroxide

Here the explicit dissociation of NaOH is considered. The drawback of this method is that we need to provide the rate of dissociation of NaOH, which is extremely fast and here assumed to be 102.

% Monoproticacid.m

%Takes 300sec to complete on 8 cores, 8 GB memory Dell machine

clear

clc

tic

Rate\_const\_no=5;

Diffeq\_no=5;

%Numerical integration

Custom\_RelTol=1e-10;

Custom\_AbsTol=1e-18;

Max\_time=1e18; %sec

% Equilibrium constants

Kw = power(10,-14);

K1 = power(10,-4);

% Initial concentration of acid

C\_acid = 1; % mol/dm^3

C\_base = 1; % mol/dm^3

% Initial volume of acid

Acid\_vol = 10; % ml

Base\_vol = 25; % ml

% Graph settings

Titration\_resolution=500; % number of titration points

dx = Base\_vol/(Titration\_resolution-1);

x = 0.0:dx:Base\_vol;

% Array for all the species

HA=size(Titration\_resolution,1);

A=size(Titration\_resolution,1);

H=size(Titration\_resolution,1);

PH=size(Titration\_resolution,1);

OH=size(Titration\_resolution,1);

FNa=size(Titration\_resolution,1);

FHA=size(Titration\_resolution,1);

% Volume and Beta

HA\_vol=size(Titration\_resolution,1);

NAOH\_vol=size(Titration\_resolution,1);

Tot\_vol=size(Titration\_resolution,1);

Beta=size(Titration\_resolution,1);

K=size(Rate\_const\_no);

Y\_init=size(Diffeq\_no);

Y\_tol=size(Diffeq\_no);

for j = 1:Diffeq\_no

Y\_tol(j)=Custom\_AbsTol;

end

outfile=fopen('Results','w');

fprintf(outfile,'Acid\_vol Base\_vol Tot\_vol HA(i,1) A(i,1) H(i,1) OH(i,1) PH(i,1) FHA(i,1) FNa(i,1) Beta(i,1)\n');

for i = 1:Titration\_resolution

% The total volume

vol\_tot = Acid\_vol + x(i);

% Concentrations of acid and base

C\_a0 = (C\_acid \* Acid\_vol) / vol\_tot;

C\_b0 = (C\_base \* x(i)) / vol\_tot ;

HA\_vol(i,1)=Acid\_vol;

NAOH\_vol(i,1)=x(i);

Tot\_vol(i,1)=vol\_tot;

% Compute the concentration of hydrogen ions

Y\_init(1)=C\_a0;

Y\_init(2)=0;

Y\_init(3)=C\_b0;

Y\_init(4)=power(10,-7);

Y\_init(5)=power(10,-7);

options = odeset('RelTol',Custom\_RelTol,'AbsTol',Y\_tol);

K(1)=K1; %power(10,pKa1);

K(2)=power(10,0);

K(3)=power(10,2); % k2;

K(4)=Kw;

K(5)=power(10,0);

[T,Y] = ode15s(@Monoprotic\_ODE,[0,Max\_time],Y\_init,options,K);

S=size(Y);

% Calculate all other species

HA(i,1) = Y(S(1),1);

A(i,1) = Y(S(1),2);

H(i,1) = Y(S(1),4);

OH(i,1) = Y(S(1),5);

FHA(i,1)=C\_a0;

FNa(i,1)=C\_b0;

% Calculate the pH value

PH(i,1) = -log10(H(i,1));

if(i>1)

Beta(i,1)=(FNa(i,1)-FNa(i-1,1))/(PH(i,1)-PH(i-1,1));

else

Beta(i,1)=0;

end

fprintf(outfile,'%.12f %.12f %.12f %.12f %.12f %.12f %.20f %.20f %.12f %.12f %.12f\n',HA\_vol(i,1),x(i), Tot\_vol(i,1), HA(i,1),A(i,1),H(i,1), OH(i,1), PH(i,1),FHA(i,1),FNa(i,1),Beta(i,1));

i

end

fclose(outfile);

% Plot it

figure();

plot(x,PH,'red')

xlabel('Volume Alkali (ml)')

ylabel('pH')

title('Titration (Volume of alkali vs pH)')

toc

This program calls the function “Monoprotic\_ODE.m”, which is defined as,

function dy = Monoprotic\_ODE(t,y,K)

dy = zeros(5,1); % a column vector

dy(1) = -K(1)\*y(1) +K(2)\*y(2)\*y(4);

dy(2) = +K(1)\*y(1) -K(2)\*y(2)\*y(4);

dy(3) = -K(3)\*y(3);

dy(4) = K(1)\*y(1) -K(2)\*y(2)\*y(4)+K(4)-K(5)\*y(4)\*y(5);

dy(5) = K(4)-K(5)\*y(4)\*y(5)+K(3)\*y(3);

# Curve fitting using Dynamic approach for phosphate buffer

We provide here the explicit matlab code to fit the experimental data of phosphate data mentioned in table 1 of the manuscript using dynamic approach. The main program ‘**curve\_fit.m**’ requires two additional script files namely, ‘**model\_func.m**’ and ‘**Phosphate\_ODE.m**’. Additionally, two input files, namely, ‘**Data**’ and ‘**Limits**’ which contains the data in x,y format and limits for fit parameters, respectively, are needed by the program. To run the program, copy all the matlab scripts and the input files and issue the command, Curve\_fit, in a matlab command window. The contents of the individual files are as follows,

# curve\_fit.m

% It takes 791 sec to complete 100 pts; 364 for 45 pts

clear;

clc;

tic

%Cluster property

processor\_no=8;

Cluster\_property = parcluster;

Cluster\_property.NumWorkers = 8;

saveProfile(Cluster\_property);

% Standard parameters

%--------------------

param\_search=1; %If exhaustive search for parameters is needed give 1, else give 0.

criteria\_res\_norm=1e-3; % Minimum squared sum of residuals

% Read the experimental data from 'Data'

%---------------------------------------

Data\_exp=fopen('Data','r');

exp\_data=fscanf(Data\_exp,'%f',[3 inf]);

fclose(Data\_exp);

exp\_data=exp\_data';

[data\_row,data\_col]=size(exp\_data);

% Read the limits for parameters

%-------------------------------

param\_limits\_ptr=fopen('Limits','r');

param\_limits=fscanf(param\_limits\_ptr,'%f',[4 inf]);

fclose(param\_limits\_ptr);

param\_limits=param\_limits';

[param\_row,param\_col]=size(param\_limits);

% Partition the parameters

%-------------------------

Max\_Partition\_no=max(param\_limits(:,4));

Partition\_array=zeros(param\_row,Max\_Partition\_no);

A=zeros(param\_row,Max\_Partition\_no);

B=zeros(param\_row,Max\_Partition\_no\*2);

limit\_init=zeros(1,1);

for i=1:param\_row

lower\_limit=param\_limits(i,1);

upper\_limit=param\_limits(i,2);

Partition\_no=param\_limits(i,4);

limit\_difference=upper\_limit-lower\_limit;

limit\_init=1;

if limit\_difference==1

limit\_difference=2;

end

r=exp((1/(Partition\_no-1))\*log(abs(limit\_difference)/limit\_init));

for j=1:Partition\_no

A(i,j)=1\*power(r,j-1);

end

if(param\_limits(i,3)==1)

for j=1:Partition\_no

B(i,j)= lower\_limit+(upper\_limit-lower\_limit)\*(A(i,j)-A(i,1))/(A(i,Partition\_no)-A(i,1));

end

param\_limits(i,4)=Partition\_no;

elseif (param\_limits(i,3)==-1)

for j=1:Partition\_no

k=Partition\_no-j+1;

B(i,j)= -(lower\_limit+(upper\_limit-lower\_limit)\*(A(i,k)-A(i,1))/(A(i,Partition\_no)-A(i,1)));

end

param\_limits(i,4)=Partition\_no;

elseif (param\_limits(i,3)==0)

for j=1:Partition\_no

k=Partition\_no-j+1;

B(i,j)= -(lower\_limit+(upper\_limit-lower\_limit)\*(A(i,k)-A(i,1))/(A(i,Partition\_no)-A(i,1)));

end

for j=1:Partition\_no-1

B(i,j+Partition\_no)= lower\_limit+(upper\_limit-lower\_limit)\*(A(i,j+1)-A(i,1))/(A(i,Partition\_no)-A(i,1));

end

param\_limits(i,4)=Partition\_no\*2-1;

end

end

Partition\_product=1;

for i=1:param\_row

Partition\_product=Partition\_product\*(param\_limits(i,4)-1)

for j=1:param\_limits(i,4)-1

%Partition\_array(i,j)=(B(i,j+1)-B(i,j))\*rand(1) + B(i,j);

Partition\_array(i,j)=(B(i,j+1)-B(i,j))\*0.5 + B(i,j);

end

end

Partition\_index=fullfact((param\_limits(:,4)-1)');

[Partition\_row,Partition\_col]=size(Partition\_index);

Final\_partition\_array=zeros(Partition\_row,Partition\_col);

for i=1:Partition\_row

for j=1:Partition\_col

Final\_partition\_array(i,j)=Partition\_array(j,Partition\_index(i,j));

end

end

min\_res\_norm=zeros(1);

min\_beta=zeros(1,param\_row);

beta\_resnorm=zeros(1,param\_row);

beta=zeros(1,param\_row);

resnorm=1e6; %resnorm=1e10;

k=0;

x = exp\_data(:,:);

y = exp\_data(:,3);

% Start the loop for all possible combinations of parameters

for i=1:Partition\_row

if param\_search==1

x\_init=Final\_partition\_array(i,:);

elseif param\_search~=1

lower\_limit=param\_limits(:,1);

upper\_limit=param\_limits(:,2);

x\_init=(upper\_limit-lower\_limit).\*rand(param\_row,1) + lower\_limit;

x\_init=x\_init';

end

matlabpool ('open',processor\_no);

options = optimoptions('lsqcurvefit','Algorithm','levenberg-marquardt','MaxFunEvals',1000,'TolFun',1e-8,'TolX',1e-8);

function\_calc = @(C,x)model\_func(C,x);

matlabpool close;

Err\_test=0;

try

[beta,resnorm] = lsqcurvefit(function\_calc,x\_init,x,y,[],[],options);

catch err

Err\_test=1;

end

if Err\_test==0

%[beta,resnorm] = lsqcurvefit(function\_calc,x\_init,x,y,[],[],options);

if k==0

min\_res\_norm=resnorm;

min\_beta=beta;

k=1;

elseif (k~=0) && (resnorm<min\_res\_norm)

min\_res\_norm=resnorm;

min\_beta=beta;

end

end

if min\_res\_norm<=criteria\_res\_norm

break;

end

disp(i);

end

%loop ends

%%%%% Standarad error and the confidence interval %%%%%%

Max\_iter=300; % 1000

Min\_Tolerance=power(10,-10); % power(10,-10)

rng(0,'twister');

opts = statset('nlinfit');

opts.RobustWgtFun = 'bisquare';

opts.MaxIter = Max\_iter;

opts.TolFun = Min\_Tolerance;

beta=min\_beta;

matlabpool ('open',processor\_no);

model\_func\_calc=@(beta,x)model\_func(beta,x);

[beta\_1,residual,jacobian,CovB] = nlinfit(x,y,model\_func\_calc,beta,opts)

matlabpool close;

SE=sqrt(diag(CovB));

ci = nlparci(beta\_1,residual,'covar',CovB);

matlabpool ('open',processor\_no);

model\_func\_calc=@(beta\_1,x)model\_func(beta\_1,x);

[ypred,delta] = nlpredci(model\_func\_calc,x,beta,residual,'Jacobian',jacobian);

lower = ypred - delta;

upper = ypred + delta;

matlabpool close;

figure()

plot(x(:,2)./x(:,1),y,'ko') % observed data

hold on

plot(x(:,2)./x(:,1),ypred,'b-','LineWidth',1.5)

plot(x(:,2)./x(:,1),[lower,upper],'r--','LineWidth',1.5)

%%%% Print the output %%%%%%

outfile=fopen('Results\_out','w');

[row\_out,col\_out]=size(beta\_1);

fprintf(outfile,'The fitted parameters with standard error and confidence interval at 95%% level (alpha=5%%)\n');

fprintf(outfile,'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n');

for i=1:col\_out

fprintf(outfile,'%f\t+/-\t%f\t[%f to %f]\n',beta\_1(1,i),SE(i,1),ci(i,1),ci(i,2));

end

[row\_out,col\_out]=size(ypred);

fprintf(outfile,'\n\n\nThe fitted data and its confidence interval at 95%% level (alpha=5%%)\n');

fprintf(outfile,'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n');

fprintf(outfile,'x\_expt\t y\_exp\ty\_pred\t[y\_ci\_lower\ty\_ci\_upper]\n');

fprintf(outfile,'---------------------------------------------------\n\n');

for i=1:row\_out

fprintf(outfile,'%f\t%f\t%f\t%f [%f to %f]\n',x(i,1),x(i,2),y(i,1),ypred(i,1),lower(i,1),upper(i,1));

end

fclose(outfile);

toc;

# model\_func.m

function z=model\_func(a,x);

[row,col]=size(x);

Rate\_const\_no=10;

Diffeq\_no=8;

%Numerical integration

Custom\_RelTol=1e-10;

Custom\_AbsTol=1e-18;

Max\_time=1e18; %sec

% Equilibrium constants

KW = power(10,-14);

K1 = power(10,a(1)); %power(10,-2.15);

K2 = power(10,a(2)); %power(10,-6.86);

K3 = power(10,a(3)); %power(10,-12.32);

% Initial concentration of acid

C\_acid = 1; % mol/dm^3

C\_base = 1; % mol/dm^3

% Initial volume of acid

% Integral specific

K=size(Rate\_const\_no);

Y\_init=size(row,Diffeq\_no);

Y\_tol=size(Diffeq\_no);

for j = 1:Diffeq\_no

Y\_tol(j)=Custom\_AbsTol;

end

for i = 1:row

% The total volume

Acid\_vol = x(i,1); % ml

vol\_tot = Acid\_vol + x(i,2);

% Concentrations of acid and base

C\_a0 = (C\_acid \* Acid\_vol) / vol\_tot;

C\_b0 = (C\_base \* x(i,2)) / vol\_tot ;

Y\_init(i,1)=0;

Y\_init(i,2)=0;

Y\_init(i,3)=0;

Y\_init(i,4)=0;

Y\_init(i,5)=C\_a0;

Y\_init(i,6)=C\_b0;

Y\_init(i,7)=power(10,-7);

Y\_init(i,8)=power(10,-7);

FHA(i,1)=C\_a0;

FNa(i,1)=C\_b0;

end

options = odeset('RelTol',Custom\_RelTol,'AbsTol',Y\_tol);

K(1)=K1; %1.7378\*power(10,-5);

K(2)=power(10,0);

K(3)=K2; %6.3\*power(10,-1);

K(4)=power(10,0); %1.7378\*power(10,-5);

K(5)=K3; %6.3\*power(10,-1);

K(6)=power(10,0); %1.7378\*power(10,-5);

K(7)=power(10,2);

K(8)=power(10,2);

K(9)=power(10,-14);

K(10)=power(10,0);

parfor i = 1:row

[T,Y] = ode15s(@Phosphate\_ODE,[0,Max\_time],Y\_init(i,:),options,K);

S=size(Y);

% Calculate the pH value

z(i,1) = -log10(Y(S(1),7));

end

# phosphate\_ode.m

function dy = Phosphate\_ODE(t,y,K)

dy = zeros(8,1); % a column vector

dy(1) = -K(1)\*y(1)+K(2)\*y(2)\*y(7);

dy(2) = K(1)\*y(1)-K(2)\*y(2)\*y(7)-K(3)\*y(2)+K(4)\*y(3)\*y(7)+K(7)\*y(5);

dy(3) = K(3)\*y(2)-K(4)\*y(3)\*y(7)-K(5)\*y(3)+K(6)\*y(4)\*y(7)+K(8)\*y(6);

dy(4) = K(5)\*y(3)-K(6)\*y(4)\*y(7);

dy(5) = -K(7)\*y(5);

dy(6) = -K(8)\*y(6);

dy(7) = K(1)\*y(1)-K(2)\*y(2)\*y(7)+K(3)\*y(2)-K(4)\*y(3)\*y(7)+K(5)\*y(3)-K(6)\*y(4)\*y(7)+K(9)-K(10)\*y(7)\*y(8);

dy(8) = K(9)-K(10)\*y(7)\*y(8);

# Inputfile: ‘Data’

877 123 6

850 150 6.1

815 185 6.2

775 225 6.3

735 265 6.4

685 315 6.5

625 375 6.6

565 435 6.7

510 490 6.8

450 550 6.9

390 610 7

330 670 7.1

280 720 7.2

# Inputfile: ‘Limits’

-2.5 -1.5 0 2

-7.5 -6 0 2

-13.5 -11 0 2

# Details on the format of the input files:

limits:

-------

Each row corresponds to one parameter to fit. Specify the lower and upper bound for each parameter. Tell whether the parameter should be positive, negative or both by setting 1,-1, or 0. Finally set the number of geometric partition to be made within this range while performing nonlinear fit.

lower\_limit upper\_limit 0:both(+/-),-1:neg,+1:positive 5: number\_of\_partitions\_in\_parameters

-2.5 -1.5 0 2

Data:

-----

X1 X2 y

877 123 6

850 150 6.1

815 185 6.2

model\_func:

-----------

z=model\_func(a,x);

a: parameters to fit as a row vector

x: single x value

z: back calculated y value