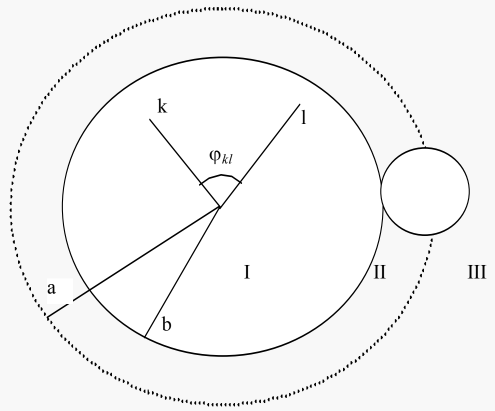
Principal formalism of PROTEIN ELECTROSTATICS

J. Kirkwood, Ch. Tanford* – *DIELECTRIC CAVITY



$$W_{el} = \frac{e^2}{2b}\sum_{k=1}^{N}\gamma_k^2\left(A_{kk} - B_{kk}\right) + W$$

$$W = \frac{e^2}{2b}\sum_{k=1}^{N}\sum_{l\neq k}\gamma_k\gamma_l\left(A_{kl} - \overset{?}{B_{kl}}\right) - \frac{e^2}{2a}\sum_{k=1}^{N}\sum_{l=1}^{N}\gamma_k\gamma_l C_{kl}$$

$$A_{kl} = 1/\varepsilon_i\left[2d_{kl}\left(1-\cos\varphi_{kl}\right)\right]^{1/2}$$

$$C_{kl} = \frac{1}{\varepsilon}\frac{\kappa a}{1+\kappa a}$$

I , II: $\Delta^2\phi = 0$

III: $\Delta^2\phi = \kappa^2\phi$

$$W_{el} = \frac{Z^2e^2}{2\varepsilon b}\left(1 - \frac{\kappa b}{1+\kappa a}\right)$$

$$\mathrm{p}K_{a,i}(k) = \mathrm{p}K_{\mathrm{int},i} - \sum_{j=1,\, j\neq i}^{N} q_j(k) W_{ij} \;/(2.3RT)$$

$$\boxed{\mathbf{SA_{ij}}}$$

$$\mathrm{p}K_{a,i} = \mathrm{p}K_{\mathrm{int},i} - \sum_{j\neq i}^{N} q_j W_{ij}\left(1 - \langle SA_{ij}\rangle\right)/(2.3RT)$$

Many different approaches

Distance-dependent dielectric "constant"

$$\boxed{d\varepsilon/dr = \lambda(\varepsilon - A)(\varepsilon_0 - \varepsilon)}$$

$$\phi(\mathbf{r}_i) = \sum_{j \neq i} q_i / \varepsilon(r_{ij})\, r_{ij}$$

$$\varepsilon(r) = A + B/[1 + k.\exp(-\lambda B.r)]$$

$$\boxed{N_A \Delta W = 2.3 RT \Delta pK_a}$$

Generalized Born

$$\Delta G_1 = -\left[\sum_i \Delta G_i^{sol,w} + \Delta G_p^{sol,w}(q=0)\right] \mathrm{x} \left(\frac{1}{\varepsilon_{in}} - \frac{1}{\varepsilon_w}\right)$$

FDPB Monte Carlo simulation

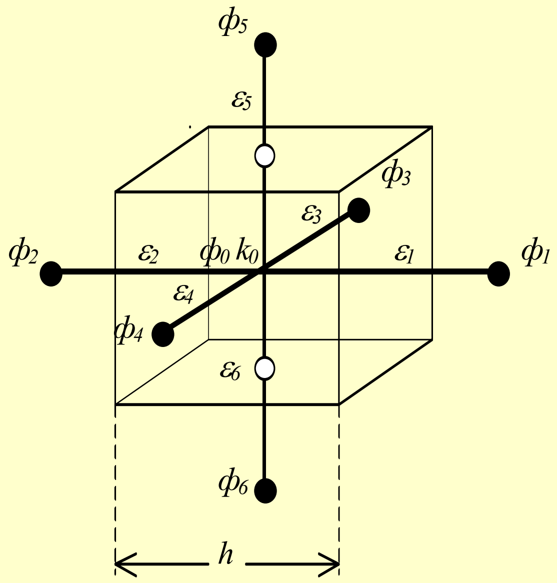
$$\phi = \sum_i^n M_i \phi_i$$

$$\theta_i = \frac{\sum_{\{\mathbf{x}\}} x_i e^{-\Delta G(\mathbf{x})/RT - \nu(\mathbf{x}) 2.3 pH}}{\sum_{\{\mathbf{x}\}} e^{-\Delta G(\mathbf{x})/RT - \nu(\mathbf{x}) 2.3 pH}}$$

$$\Delta G(\mathbf{x}) = \sum_i^{\mathbf{N}} \Delta G_{\mathbf{i}}^{\mathrm{int}} x_i + \frac{1}{2} \sum_{i,j}^{N} W_{ij} (q_i^0 + x_i)(q_j^0 + x_j)$$

Linear and non-linear Poisson-Boltzmann Equation

$$\nabla \bullet [\varepsilon(\mathbf{r}) \nabla \bullet \phi(\mathbf{r})] - \bar{\kappa}(\mathbf{r})^2 \sinh[\phi(\mathbf{r})] + 4\pi\rho(\mathbf{r}) = 0$$



$\Delta G = - 2.3\ RT\ \lg K$; $pK = - \lg K$;

$$\Delta pK^{solv}_{ai} = -\frac{1}{2.3RT}\left[\frac{q_i(\phi_{i,prot}(\mathbf{r}_i) - \phi_{i,\mathrm{mod}}(\mathbf{r}_i))}{2}\right]$$

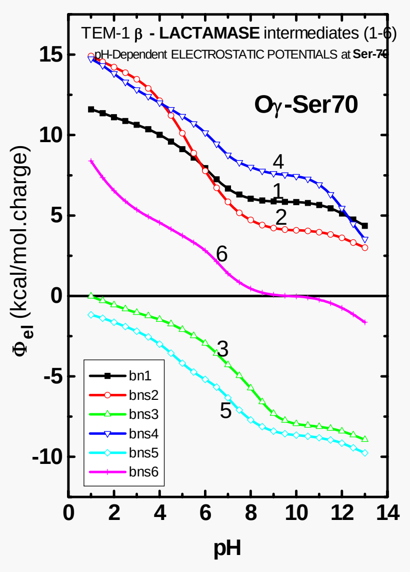
$$\Delta\, pK^{pair}_{a,i} = -\frac{1}{2.3RT}\sum_{j \neq i}^{N} q_j(\mathrm{pH}).\phi_i(r_j)$$

Local and Global EI property: ELECTROSTATIC POTENTIAL (EP)

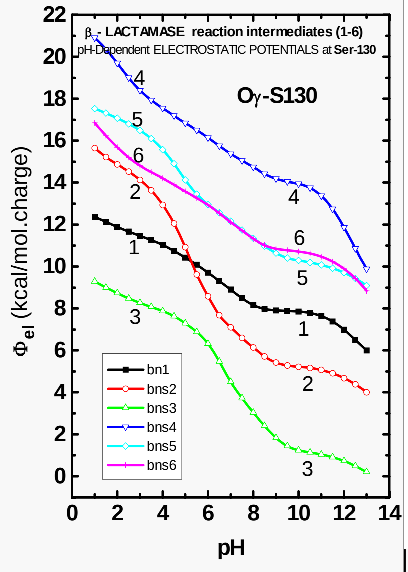
Site EP

$$\nabla \cdot (\varepsilon(r)\nabla\varphi(r)) - \kappa^2\varepsilon_s\varphi(r) + 4\pi\rho_p(r) = 0$$

$$\Phi_{el,i}\,(\mathrm{pH}) = 2.3\ RT \sum_i \Delta pK_i\,(\mathrm{pH}) \sum_{i \neq j} Z_{ij}\,(\mathrm{pH})\,(1 - SA_{ij})$$



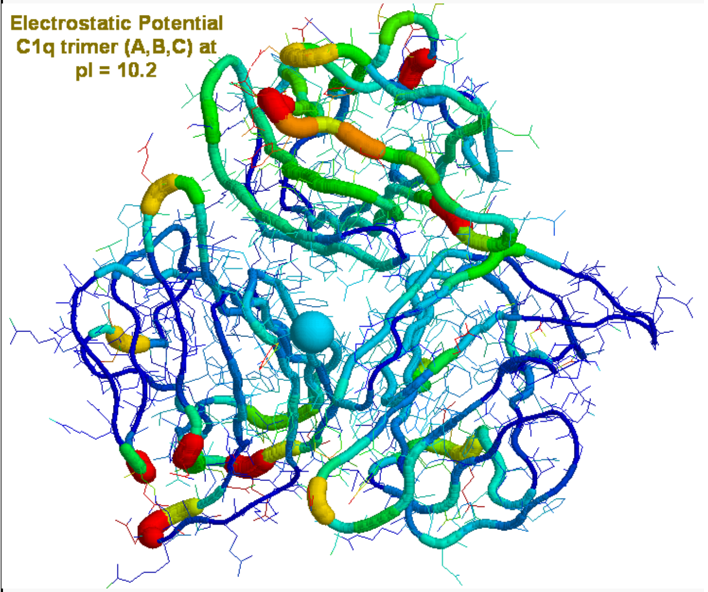
S70-pot.org



S130-pot.org

Protein EP

$$\varphi(i) = \frac{\sum_j \varepsilon(ij)\varphi(j) + \dfrac{4\pi q(i)}{h}}{\sum_j \varepsilon(ij) + \kappa^2 h^2 \varepsilon_s}$$



Electrostatic Potential (EP) Derived Electric Moments (Haskell – numerical linear algebra Lapack library interface)

Whatever method for computation of electrostatic potential grid is chosen all procedures share common basis and inherit analogous problems in the subsequent electric moment calculation. In any case numerical difficulties emerge that might make it impossible to perform the fitting unequivocally – i.e. resultant electric moment scalar value and vector orientations turn out to be ambiguous. Singular value decomposition (SVD) comes at a rescue**:** $\boldsymbol{A = U S V^*}$ where **U** and **V** are unitary (orthonormal) matrices, $\boldsymbol{V^*}$ is the conjugate transpose of **V** and **S** is diagonal whose elements are the singular values of the original matrix. The separable form turns to be useful for certain class of problems: $A = \sum_j \mathbf{H}_j = \sum_j \zeta_j U_j \times V_j$, ζ_j being ordered singular values. It can be proved that no rank-deficiency problems are encountered if the least-squares fit is performed using pseudoinverses calculated by singular value decomposition. The pseudo inverse $\boldsymbol{\Omega}^+$ of the matrix $\boldsymbol{\Omega}$ with singular value decomposition: $\boldsymbol{\Omega = U \Sigma V^*}$, as a special case – in eigenvalue decomposition form:

$$\boldsymbol{\Omega^*\Omega = V\, \Sigma^*\, U^*\, U\, \Sigma\, V^* = V\, (\Sigma^*\, \Sigma)\, V^*} \text{ or } \boldsymbol{\Omega\,\Omega^* = U\, \Sigma\, V^*\, V\, \Sigma^*\, U^* = U\, (\Sigma\, \Sigma^*)\, U^*}$$

is represented by the following matrix expression:

$$\boldsymbol{\Omega^+ = V\, \Sigma^+\, U^*},$$

where $\boldsymbol{\Sigma}^+$ is the transpose of $\boldsymbol{\Sigma}$ with every nonzero entry replaced by its reciprocal. The pseudo inverse is at the heart of state of the art algorithms to solve linear least squares problems.

1. Example SVD Haskell code in interfacing LAPACK routine dgelss pseudoinverse of a matrix

```
pinv :: Field t => Matrix t -> Matrix t
pinv m = linearSolveSVD m (ident (rows m))
```

The intermediate SVD step is a two stage procedure. At first a Householder reflections is performed to reduce matrix to bidiagonal form. Then a variant of orthogonal decomposition - the QR algorithm is applied:

2. Example SVD Haskell code in interfacing LAPACK routine dgeqr2 - QR factorization matrix - q is unitary and r is upper triangular.

```
qr            :: Matrix t -> (Matrix t, Matrix t)
```

3. Example SVD Haskell code in interfacing LAPACK routines that diagonalize a matrix and find singular values.

```
full :: Element t
     => (Matrix t -> (Matrix t, Vector Double, Matrix t)) -> Matrix t -> (Matrix t,
                     Matrix Double, Matrix t)
full svd' m = (u, d ,v) where
    (u,s,v) = svd' m
    d = diagRect s r c
    r = rows m
    c = cols m
```

4. Example SVD Haskell code in interfacing LAPACK routines that return singular values as well as orthogonal matrices

```
nonzero :: Element t
        => (Matrix t -> (Matrix t, Vector Double, Matrix t)) -> Matrix t -> (Matrix t,
                     Vector Double, Matrix t)
economy svd' m = (u', subVector 0 d s, v') where
    (u,s,v) = svd' m
    sl@(g:_) = toList s
    s' = fromList . filter (>tol) $ sl
    t = 1
    tol = (fromIntegral (max r c) * g * t * eps)
    r = rows m
    c = cols m
    d = dim s'
    u' = takeColumns d u
    v' = takeColumns d v
```

Motivation behind *HASKELL-LAPACK* interface approach

Numerical operations on vectors and matrices have very fast implementations in linear algebra libraries such as **ATLAS** (BLAS) and **LAPACK**. Major improvements in efficiency have origin in advanced use of cache and application of specialized processor instructions. **Haskell** call of an operation such as matrix multiplication using these libraries may execute orders of magnitude faster than a straightforward low-level implementation in **C/C++** or **FORTRAN** programming languages(1,2,3). Hence our motivation to use schemes of combined efficiency (advanced computational linear algebra libraries – **C/C++** or **FORTRAN**) and expressivity – **Haskell**. Whenever one employs matrix algorithms errors are found at run-time or statically at the level of tensor rank/element type (3). For software, like **PHEMTO** underlying computational engine, implementing procedures with heavy numerical code, a method that allows static determination of ranks is needed and the compromise is achieved via **Haskell – LAPACK** interface.

Reference and Internet resources:

1.Ruiz, A. Matrix computations in haskell based on the gsl. (2005) http://dis.um.es/_alberto/GSLHaskell/matrix.pdf

2. Kiselyov,O. and Shan,C. (2004) Functional pearl: implicit configurations–or,type classes reflect the values of types. In Haskell '04: Proceedings of the ACM SIGPLAN workshop on Haskell, pages 33–44, NewYork, NY, USA, ACM Press

3. Eaton,F. (2006) Statically typed linear algebra in haskell (papers and source code). http://ofb.net/_frederik/stla/