NON-LINEAR LEAST SQUARES Pseudocode

① function perform_Nonlinear_Least_Squared

- initialize:
  - $\vec{\beta}^p$ = initial guess (3×1 column vector) [1 0.5 0.3]
  - error
  - N (# iterations)
  - tolerance (1e-5)
- read in data $\{(x_j, y_j)\}_{j=1}^{N}$
  - create xVec & yVec

while err > tol

2 different functions

→ Form "Jacobian-like" matrix, J, for $\vec{\beta}^p$ and data (function ②)

→ Form $\Delta\vec{y}$ for $\vec{\beta}^p$ and data (function

→ form pseudo-inverse and get:

$$\Delta\vec{\beta}^k = (J^T J)^{-1} J^T \Delta y^k$$

$\vec{\beta}^N = \vec{\beta}^p + \Delta\vec{\beta}^k$ % New approx

$err = \|\vec{\beta}^N - \vec{\beta}^p\|_2$ % compute error (l^2-Norm)

$\vec{\beta}^p = \vec{\beta}^N$ % redefine for next iter

$N = N+1$ % update # of iters

end

② function J=compute_Jacobian(
- input ($\vec{\beta}^p$, xVec)
- output J

```
for i = 1:length(xVec)   % iterate thru data pts
    J(i,1) = f_B1(xVec(i), β^k)
        % partial of f WRT B1
    J(i,2) = f_B2(xVec(i), β^k)
        % partial of f WRT B2
    J(i,3) = f_B3(xVec(i), β^k)
        % partial of f WRT B3
end
```

computing partials:

$$f(x;\vec{\beta}) = \beta_1 e^{\frac{-(x-\beta_2)^2}{2\beta_3^2}}$$

$$\frac{\partial f}{\partial \beta_1}(x;\vec{\beta}) = e^{\frac{-(x-\beta_2)^2}{2\beta_3^2}}$$

$$\frac{\partial f}{\partial \beta_2}(x;\vec{\beta}) = \frac{\beta_1(x-\beta_2)}{\beta_3^2} e^{\frac{-(x-\beta_2)^2}{2\beta_3^2}}$$

$$\frac{\partial f}{\partial \beta_3}(x;\vec{\beta}) = \frac{\beta_1(x-\beta_2)^2}{\beta_3^3} e^{\frac{-(x-\beta_2)^2}{2\beta_3^2}}$$

③ function dely = compute_dely()

- input ($\vec{B}^P$, xVec, yVec)
- output dely

```
for i = 1 : length(yVec)
    dely(i) = yVec(i) - f(xVec(i); B^P)
        % where f = B1 e^(-(x-B2)^2 / 2B3^2)
end
```