To recap, in Part 1 we defined the model form $y = w_0 + \langle w, x\rangle - e$ as well as the associated loss function $L_2 = \sum_i e_i^2 = \langle e, e\rangle$. Part 2 discussed the method of gradient descent, expressing the gradient as a function of parameter estimates $\nabla_w L_2(w) = (2\langle \mathbf{1}, e\rangle, 2\langle x_1, e\rangle, \ldots, 2\langle x_n, e\rangle)$, as well as an adaptive learning method for optimization.

For implementation in software, we are using the python scripting language, as well as associated packages. In particular, the numpy package forms the back-bone of storing arrays in a convenient form for calculations. To aid in this implementation, the model is represented in a more matrix, friendly form. Taking the number of entries in the data to be d, we create the following design matrix

$$\begin{array}{c|ccccc} & \multicolumn{5}{c}{\textit{corresponding weight}} \\ & b & w_1 & w_2 & \cdots & w_n \\ \textit{example} & & & & & \\ 1 & 1 & x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ 2 & 1 & x_{1,2} & x_{2,2} & \cdots & x_{n,2} \\ 3 & 1 & x_{1,3} & x_{2,3} & \cdots & x_{n,3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ d & 1 & x_{1,d} & x_{2,d} & \cdots & x_{n,d} \end{array} \rightarrow X_{d\times(n+1)},$$

which allows simple matrix multiplication to find the prediction and the gradient.

$y + e = f(x; w) = \langle X, w\rangle$

$\nabla_w L_2(w) = (2\langle \mathbf{1}, e\rangle, 2\langle x_1, e\rangle, \ldots, 2\langle x_n, e\rangle) = (2\langle e, \mathbf{1}\rangle, 2\langle e, x_1\rangle, \ldots, 2\langle e, x_n\rangle) = 2\langle e, X\rangle.$

Here follows an account of the proposed solution algorithm in python-based pseudocode.

```
# Bring in data
        data = import(dataFile)

# Clean the data
        # Standardize variables
        labelStd, featureStd = standardize(label, feature)
        |
        *Do Stuff*
        |
        dataClean = ...
```

```
# Run Regression
    # Initialized Model Variables
    weights = array([1,1])
    error = dot(design, weights)-labelStd
    loss = ssq(error)

    # Initialized Algorithm Variables
    tolerance = 1  # Cutoff for change in loss    # aka, trade-off between acc. and comp.
    learn = 0.5  # to be adjusted for speed of learning / convergence
    accel = 0.3  # weight given to previous update (may need adjustment)
    update = learn*dot(error, design)
    weights = weights - update
    loss_last = 0
    iter = 0

    while abs(loss-loss_last) > tolerance and iter < 1000:
        loss_last = loss
        error = dot(design, weights-accel*update)-labelStd   # see “Recall”
        loss = ssq(error)
        update = accel*update + learn*dot(error, design)  # see “Recall”
        weights = weights – update
        iter += 1
    print(iter, loss)
    print(weights)

    *Do stuff to create model function*
    |
    model = ...

    # Recall that the adaptive update algorithm is $u_k = hu_{k-1} + r \cdot \nabla_w L_2(w_{k-1} - hu_{k-1})$
    # here h is “accel” and r is “learn”. To get $\nabla_w L_2(w_{k-1} - hu_{k-1})$,
    # we need errors as a function of $w_{k-1} - hu_{k-1}$.
    # Note that “Ordinary” gradient descent is recovered by setting accel = 0.

# Predict Values
    # import
    dataTest = import(testFile)

    # Extract features
    features  = dataTest[...

    # Predict
    predictions = model(features)
```