CZ3106 Symbolic Computing, Lab 3

Dr. Wang Jian-Sheng   For the weeks 10-22 Feb, due Saturday 22 Feb 03

1. Finish the code for Gaussian elimination method for solving linear equations. You can copy my code for the elimination part; you need to supply your code on the `backsub[..]` for backward substitution. Test your program with at least one numerical input, and one symbolic input.

2. Define a least-square fit function similar to the built-in function `Fit[]`. Our fitting function takes the form

   `leastsquarefit[{{`x_1`,`y_1`}, {`x_2`,`y_2`}, ...}, {`$f_1(x)$`, `$f_2(x)$`, ...}, `x`]`

   where the first argument contains a list of pair of points, the second argument is a list of linearly independent basis functions, and the last argument (a symbol) is the independent variable of the functions. A polynomial fit is obtained if we take $f_i(x) = x^i$, $i = 0, 1, \ldots, n$. The function returns the fitted result

   $$F(x) = \sum_i c_i f_i(x)$$

   such that the square deviation of error is minimized. For some more discussions, see T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms," p. 768–771. We use the result that the fitting coefficient c_i taken as a column vector is determined by the equation

   $$A^T A c = A^T y,$$

   where the matrix A is defined by $A_{ij} = f_j(x_i)$, (the dimensions of A are the number of sample points by the number of fitting functions) and A^T is the transpose of A, and y is the column vector y_i. Use the matrix and linear algebra capability of Mathematica in your definition. Given some testing sample outputs (e.g. a plot with points and fitted curve).

3. [A difficult one] (a) Write a function `buildtree[]` that takes a list and builds a binary tree with nested lists. Each node is represented as

   ```
   { nodeelement, {leftchild}, {rightchild} }
   ```

   The children in turn have similar recursive data structure representation. The order of the elements in the tree is such that all the nodes of left subtree is not late in the Mathematica internal ordering than the parent, as determined by `Sort[ ]`. The elements of the input list are used from the beginning one by one, and are added to the tree. Here are some examples,

   ```
   buildtree[{5,3,2,9}]  => {5, {3, {2, {}, {}}, {}}, {9, {}, {}}}
   buildtree[{5,c,2,a}]  => {5, {2, {}, {}}, {c, {a, {}, {}}, {}}}
   buildtree[{{a,b},{1,2},{u,v,w}}]=>{{a,b},{{1,2},{},{}},{{u,v,w},{},{}}}
   ```

   Note that the nodes themselves can be arbitrary expressions. *Hint*: Define a separate function `addelement` for adding one element to a tree. Think in terms of recursive call, but do not think in terms of pointers. There is no pointer in Mathematica.

   (b) Write a function `traverse[]` that takes a tree and builds a list in-order; that is, in an order that visit the left subtree first, then the node, and then the right subtree. `traverse[buildtree[`e`]]` produces the same result as `Sort[`e`]`. here are three examples:

   ```
   traverse[buildtree[{5,3,2,9}]]  => {2, 3, 5, 9}
   traverse[buildtree[{5,c,2,a}]]  => {2, 5, a, c}
   traverse[buildtree[{{a,b},{1,2},{u,v,w}}]] => {{1,2},{a, b},{u,v,w}}
   ```

   Some background in computer algorithms will help.