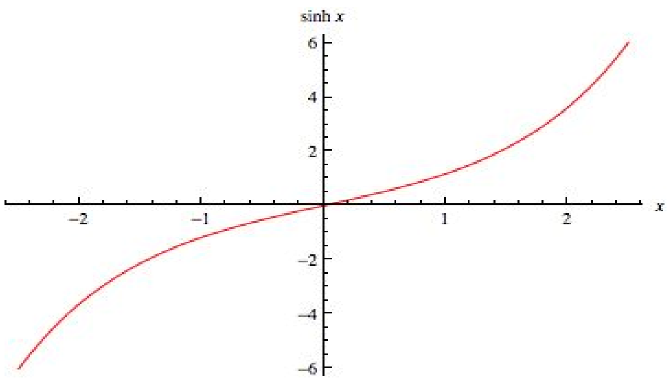
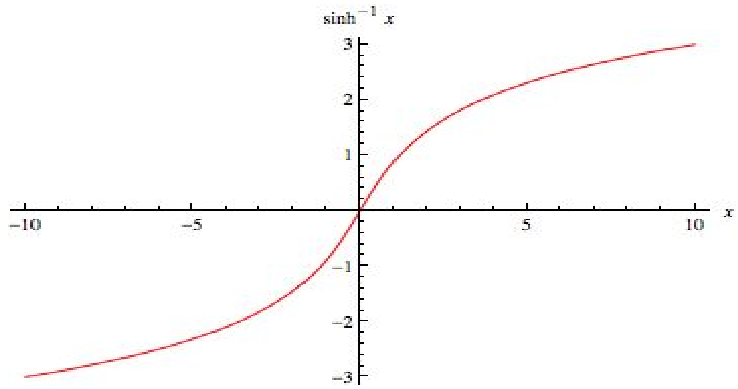
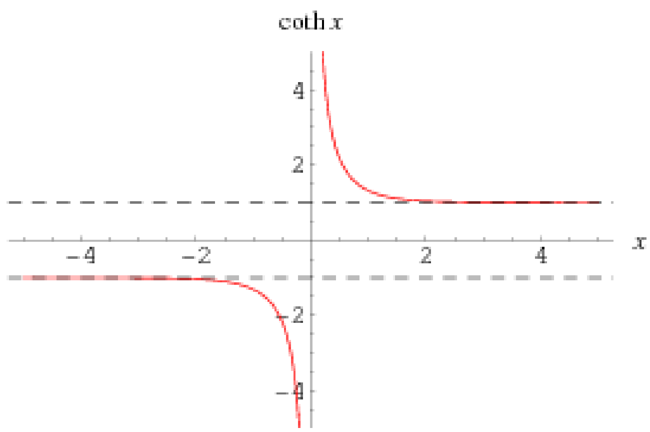
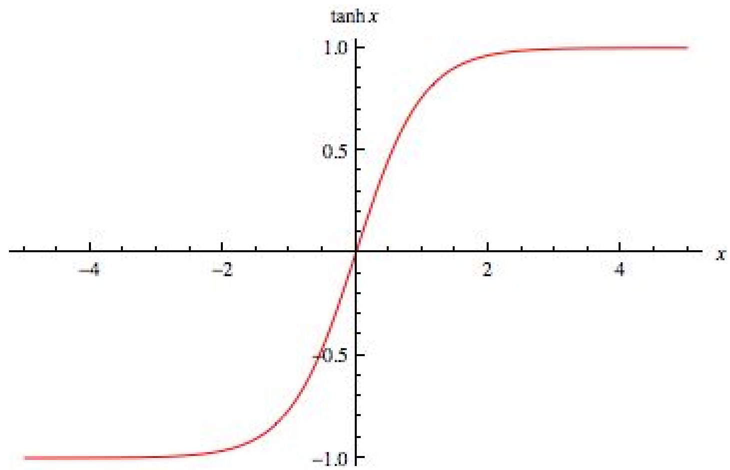
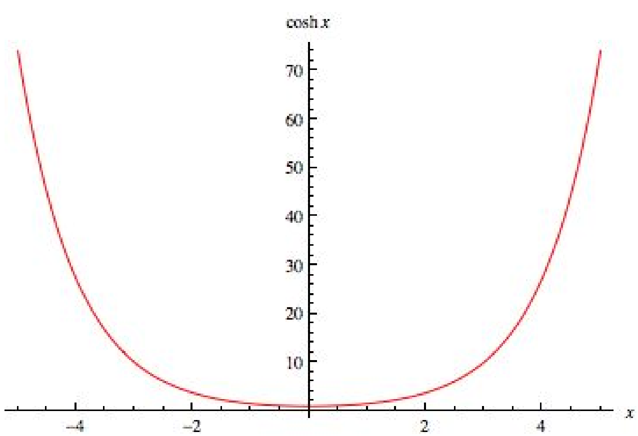
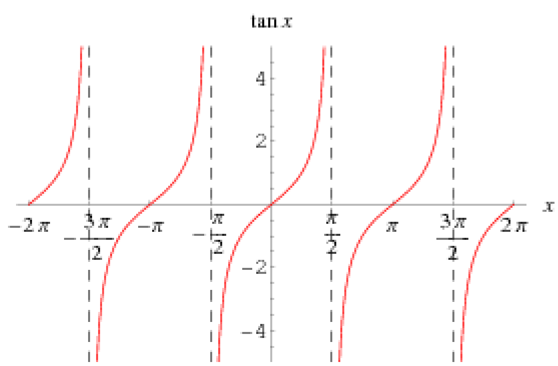

sinh x
x

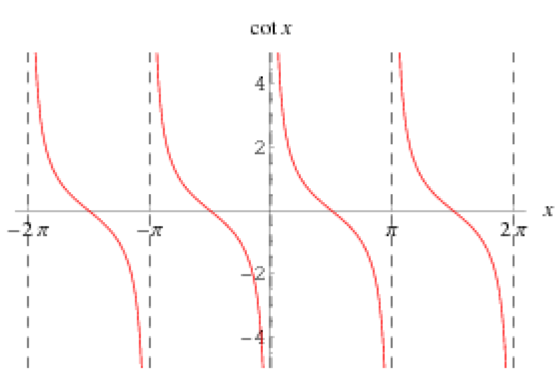

sinh⁻¹ x
x


coth x
x


tanh x
x


cosh x
x


tan x
x


cot x
x

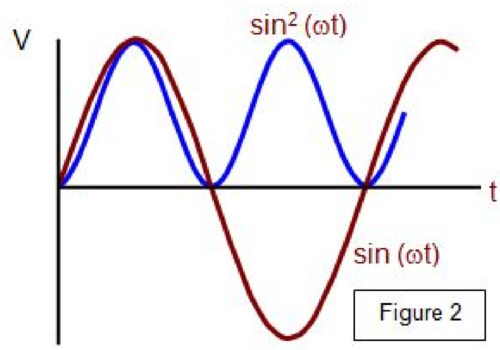

V
sin^2 (ωt)
t
sin (ωt)

Figure 2

```
## note, must be tested against something with known value, like infinite square well
'''

    matrix
        M = matrix
        Psi = vector (eigenfunctions = eigenvectors)
        E = vector (eigenvalues)
        M * Psi

    linear interpolation
        - spline interpolation integration

    normalization
        int modsquared psi_found = int |A|^2 psi_normalized = A^2 -> psi_normalized = 1/sqrt(A) psi_found


        NOTE: h_bar^2/2m is not defined

        conversion factor - x = x/a s.t. in real space potential is between 0 and a
        then eigenvalues are 1/a^2 -> but when transforming into actual energy, keep track that x_max represents a
        for potential to, when potential must be in units of energy

Note - with time evolution must choose correct order of magnitude of delta_t
'''


def build_matrix(n, delta, v_list):
    H_matrix = sp.csr_matrix((n,n));
    for diagonal_index in range(n):
        # main diagonal element
        H_matrix[diagonal_index, diagonal_index] = (2/float(delta**2)) + v_list[diagonal_index];

        # lower off diagonal
        lower_off_diagonal = diagonal_index - 1;
        #print(lower_off_diagonal);
        if(lower_off_diagonal >= 0):
            H_matrix[diagonal_index, lower_off_diagonal] = (-1)*(1/float(delta**2));

        # higher off diagonal
        higher_off_diagonal = diagonal_index + 1;
        #print(higher_off_diagonal);
        if(higher_off_diagonal < n):
            H_matrix[diagonal_index, higher_off_diagonal] = (-1)*(1/float(delta**2));

    print(H_matrix.todense())
    return H_matrix;
```