Davidson

May 24, 2018

1 Davidson Algorithm

by Iliya Sabzevari

The Davidson algorithm is an efficent way to find a number of the lowest (or highest) valued eigenvalues of a large, diagonally dominant matrix. The general goal of the algorithm is to solve the eigenvalue problem for the matrix in question in a gradually growing subspace of orthonormal vectors and hope the eigenvector that corresponds to the lowest (or highest) valued eigenvalue exists in the span of this subspace. The main steps are:

1. Guess an orthonormal basis of vectors with dimension greater than the number of eiegenvalues you are solving for
2. Calculate the residue vector and check for convergence. The residue vector is the vector left over after trying to solve the eiegenvalue problem with the guess vectors
3. Calculate the correction vector based on the residue vector and the approximate eigenvalue pairs
4. Add the correction vectors to your orthonormal subspace and repeat starting from step 2

The written function can be seen below.

```
In [1]: import math
        import numpy as np

        def Davidson(H_func,H_diag,neig):          #input Direct Method Hamiltonian, Diagonal of H
                a = 8                                  #number of vectors in initial sample space
                tol = 1e-8
                n = len(H_diag)
                t = np.eye(n,a)                          #set of test unit vectors in initial samp
                V = np.zeros((n,a))                   #array to store sample space

                for i in range(a):                    #input test vectors into sample space matrix
                        V[:,i] = t[:,i]


                theta_old = np.zeros(neig)           #initialize old and new eigenvalue guesses, "T
                theta_new = np.ones(neig)

                count = 1                           #keep track of number of iterations
```

```
        while np.linalg.norm(theta_old-theta_new) > tol:
                theta_old = theta_new        #step theta
                V,R = np.linalg.qr(V)        #use python's QR decomp. to ensure sample s
                HV = np.zeros((n,a*count))
                for i in range(a*count):
                        HV[:,i] = H_func(V[:,i])
                VHV = np.dot(V[:,:(a*count)].T,HV) #build matrix in subspace
                theta,s = np.linalg.eig(VHV)        #diagonalize
                index = np.argsort(theta)        #sort eigenvalues and eigenvectors
                theta = theta[index]
                s = s[:,index]
                V = np.c_[V,np.zeros((n,a))]        #grow sample space matrix
                for i in range(a):                 #loop through test vectors
                        test = np.dot(V[:,:(a*count)],s[:,i])        #change basis of ei
                        r = H_func(test) - theta[i]*test        #calculate residue vecto
                        q = -(1/H_diag[i] - 1/theta[i])*r        #calculate correction v
                        V[:,(i+(a*count))] = q                        #add correction ve
                theta_new = theta[:neig]                        #update guesses to eigen
                count = count + 1
        return theta_new
```

To test the algorithm, I made a test script that constructed a diagonally dominant matrix and computed the 4 lowest valued eigenvalues and compared the calculation time to the eiegenvalue solver in numpy.

```
In [6]: import numpy as np
        import Davidson as D
        import time

        #Build diagonally dominant Hamiltonian and Direct method function of Hamiltonian
        n = 1500
        H = np.zeros((n,n))


        r = range(n)
        for i in r:
                H[i,i] = i+1        #take diagonal elements to be increasing integer values
                for j in r[(i+1):]:        #take off diagonal elements to be decreasing in order
                        H[i,j] = (10**(i-j+1))
        H = (H.T + H)/2

        def A(v):                #make direct method function
                return np.dot(H,v)

        #Davidson
        start_davidson = time.time()

        E = D.Davidson(A,np.diag(H),4)
```

```
        end_davidson = time.time()

        start_numpy = time.time()

        print("Davidson = ", E,":",end_davidson-start_davidson, "seconds")

        #Numpy
        start_numpy = time.time()

        E,V = np.linalg.eig(H)
        E = np.sort(E)

        end_numpy = time.time()

        print("Numpy = ", E[:4],":",end_numpy - start_numpy, "seconds")
```

```
Davidson =  [0.78251653 1.9679405  2.99718315 3.9998414 ] : 0.030191659927368164 seconds
Numpy =  [0.78251653 1.9679405  2.99718315 3.9998414 ] : 9.659271717071533 seconds
```

Running on the virtual machine on my personal computer, the Davidson algorithm calculated the 4 lowest eigenvalues of a 1500x1500 diagonally dominant matrix with a high degree of accuracy in almost 0.03 seconds, compared to numpy in almost 10. Not to bad (if you ignore the fact numpy calculated the entire eigenvalue spectrum).