PHYS4070/7270 Worksheet: Week 11 (13/05/2021)

Part A:

Some tips of using std::vector, including using vector-of-vector.

Basics Example (refresher):
Interactive version: https://godbolt.org/z/5csjznTfc

```
#include <iostream>
#include <vector>
int main() {
  std::vector<int> v{3, 1, 4};

  std::cout << "Regular for loop:\n";
  for (int i = 0; i < v.size(); ++i) {
    std::cout << v.at(i) << "\n";
  }
  std::cout << "Ranged for loop:\n";
  for (auto x : v) {
    std::cout << x << "\n";
  }
  std::cout << "Ranged for loop, by reference:\n";
  for (auto &x : v) {
    std::cout << x << "\n";
    x *= 2;  // by reference, can modify values
  }
  std::cout << "Ranged for loop, after modifying:\n";
  for (auto x : v) {
    std::cout << x << "\n";
    x *= 2;  // by reference, can modify values
  }
}
```

std::vector is a c++ *container* - and can hold values of *any* type, including another vector. A vector-of-vector is therefore, in a sense, a 2D vector.

NOTE however, that a vector-of-vector does not store all the memory contiguously in a single block. Each internal vector stores all the elements in a single block, but each vector is stored in a different location. That is why we cannot use a vector-of-vector in place of regular C-array to pass matrix to LAPACK.

However, vector-of-vector is still very useful in many situations, since it lets us easily pass individual internal vectors around to different functions. For example, you may store a set of wavefunctions in a vector-of-vector-of-double, then you can pass individual wavefunctions around, which is very nice.

Example for vector-of-vector:
Interactive version: https://godbolt.org/z/7v67obYef

```
#include <iostream>
#include <vector>

// Simple function; prints elements of a vector of int
void printVector(const std::vector<int>& v) {
  for (auto x : v) {
    std::cout << x << ", ";
  }
  std::cout << '\n';
}

int main() {
  // Create a vector of 3 empty vectors
  std::vector<std::vector<int>> v(3);
  std::cout << v.at(0).size() << "\n";
  std::cout << v.at(1).size() << "\n";
  std::cout << v.at(2).size() << "\n";

  // Nb: Can use ranged for loop too!
  for (const auto& x : v) {
    // 'auto' evaluiates to 'std::vector<int>' here
    // By reference, avoid copying entire vector
    std::cout << x.size() << "\n";
  }

  // Can fill internal vectors
  v.at(0).push_back(3);
  v.at(0).push_back(2);
  v.at(0).push_back(1);

  // Or set them equal to new vectors
  v.at(1) = {6, 7, 8, 9, 10};

  // Can send each internal vector to function!
  printVector(v.at(1));
}
```

Operator overloads

You can make parts of your code much simpler to read/write by defining *operator overloads* for std::vector.

Consider the below example, which provides a '+' operator:

Example
Interactive version:https://godbolt.org/z/d8GP9PWsf

```
#include <cassert>
#include <iostream>
#include <vector>

// Overload for '+'
```

```
std::vector<double> operator+(std::vector<double> &a,
                              const std::vector<double> &b) {
  assert(a.size() == b.size() && "a and b must be same size");
  for (auto i = 0ul; i < b.size(); ++i) {
    a.at(i) += b.at(i);
  }
  return a;
}

// Simple function; prints elements of a vector of double
void printVector(const std::vector<double> &v) {
  for (auto x : v) {
    std::cout << x << ", ";
  }
  std::cout << '\n';
}

int main() {
  std::vector<double> v1{3.0, 3.5, 4.0};
  std::vector<double> v2{0.1, 0.2, 0.3};

  printVector(v1);
  printVector(v2);
  printVector(v1 + v2); //much easier than doing slow way!
}
```

NOTE it can be considered bad practice to overload operators for built-in types (like std::vector) - since it may conflict with other parts of a code base. For this reason, it is encouraged to wrap these overloads in a namespace. Then, use the `using namespace .....` directive to provide *local* access to the operators only within a small scope, reducing risk of conflicts (hardly important for small projects)

(For example, should '+' add together each element of a vector like {a+x, b+y, c+z}? Or join two vectors, like {a,b,c,x,y,z}? What if you want different behaviour in different cases? What if you and a co-worker disagree? In fact, this is the reason these are not already define as standard in c++)

Example

```
namespace MyVectorOverloads {
std::vector<double> operator+(std::vector<double> &a, const std::vector<double>
&b) {...}
}

int main() {
  using namespace MyVectorOverloads; //put this in as narrow a scope as
practicle!
  std::vector<double> v1{3.0, 3.5, 4.0};
  std::vector<double> v2{0.1, 0.2, 0.3};
  printVector(v1 + v2);
}
```

Part B: Worksheet tasks

In the assignment, you are tasked with Solving Schodinger equation for a many-electron atom; here we will practise the procedure for the simplest case of hydrogen.

- The radial Hamiltonian for Hydrogen atom is:

$$H_r = \frac{-1}{2}\frac{\partial^2}{\partial r^2} - \frac{Z}{r} + \frac{l(l+1)}{2r^2},$$

- We will use a new very powerful method to solve the Schrodinger equation, by expanding the solutions over a basis of B-spline (basis) functions, b. (Use provided code to calculate B-splines)

$$P(r) = \sum_j^{N_b} c_j b_j(r),$$

- Solve the Schrodinger equation for Hydrogen by solving the eigenvalue problem using DSYGV:

$$\sum_j \langle i|\hat{H}_r|j\rangle c_j = \varepsilon \sum_j \langle i|j\rangle c_j$$
$$\implies H_r\,\vec{c} = \varepsilon\, B\,\vec{c}.$$

$$H_{ij} = \langle i|\hat{H}_r|j\rangle = \int b_i(r)\hat{H}_r b_j(r)\,dr\,, \qquad B_{ij} = \langle i|j\rangle = \int b_i(r)b_j(r)\,dr,$$

You can use any integration scheme for these integrals - it will be much easier if you store the values of the B-splines in an array *before* trying to do the integrals. As described in lectures, discard the first two (index=0 and 1) B-splines, and the last one (index=n-1) to enforce the boundary conditions.

Use ~30-60 Bsplines of order k=7. You will have to choose good r0 and rmax.

1. Compare energies for s and p states to expected
    - Note: Biggest source of error likely comes from integration grid, r0, rmax, and num_stepd
    - Since the H and B matrix sizes depend on number of B-splines used, NOT number of integration points, we can increase number of points without slowing down code very much!
2. Use expansion coefficients and B-splines to construct wavefunctions; check that they are properly normalised (they should already be)
3. Plot wavefunctions for 1s, 2s, and 2p
4. Think about simple extension to this needed for assignment.

DSYGV parameters (very similar to DSYEV, can adapt previous code)

```
extern "C"
int dsygv_(
    int *ITYPE,    // =1 for problems of type Av=eBv
    char *JOBZ,    // ='V' means calculate eigenvectors
    char *UPLO,    // 'U': upper triangle of matrix is stored, 'L': lower
    int *N,        // dimension of matrix A
    double *A,     // c-style array for matrix A (ptr to array, pointer to a[0])
                   // On output, A contains matrix of eigenvectors
    int *LDA,      // For us, LDA=N
```

```
    double *B,     // c-style array for matrix B [Av=eBv]
    int *LDB,      // For us, LDB =N
    double *W,     // Array of dimension N - will hold eigenvalues
    double *WORK, // 'workspace': array of dimension LWORK
    int *LWORK,    // dimension of workspace: ~ 6*N works well
    int *INFO      // error code: 0=worked.
);
```

Example for using the provided B-spline code

```
#include "bspline.hpp"
#include <iostream>
int main(){
  double r0 = 1.0e-3;
  double rmax = 50.0;
  int k_spine = 7;          // order of B-splines
  int n_spline = 60;

  // Initialise the B-spline object
  BSpline bspl(k_spine, n_spline, r0, rmax);

  // Value of the 1st (index=0) B-spline at r=0
  std::cout << bspl.b(0, 0.0) << "\n";
  // Value of the 6th (index=5) B-spline at r=1.5 au
  std::cout << bspl.b(5, 1.5) << "\n";
  // Value of the last (index=N-1) B-spline at r=rmax
  std::cout << bspl.b(n_spline - 1, rmax) << "\n";
}
```