Summary of Research Activities Summer 2004

Justin Lambright
Geneva College
3200 College Avenue
Beaver Falls, PA 15010

To begin the summer, I reviewed some of what I had learned in the previous summer working with Joe Berg for Dr. van de Sande. Tyler Anderson had not seen the material before and was given a similar set of lectures and problems to work out as we were given last summer. By paying attention to the lectures and reworking some problems, I managed to get a nice review of what I had learned during the previous summer.

The next task I was given was to calculate some terms that would be used in the Hamiltonian matrix. First I calculated the values of $(\bar{\psi}\psi)$, $(\bar{\psi}\gamma^5\psi)$, among others. These values are part of operators in the Hamiltonian matrix. After working these values out separately, I was able to easily calculate the operators of which they were a part. $(\bar{\psi}\gamma^5\psi)^2$ was added to the meson calculation at the end of the summer of 2003, so the results of these calculations were already known. This was once again a sort of review to get me back in the mindset needed to do the new calculations. I then went on to calculate some new operators such as $(\bar{\psi}\gamma^+\psi)(\bar{\psi}\gamma^-\psi)$. These calculations required a good deal of algebra and a little calculus to finish. The intent of the calculations was to add the operators to the old code written in previous years. However, after Dr. van de Sande checked over my work and considered the results of the calculation, he decided to switch methods of calculating the Hamiltonian due to the difficulty of implementing these new operators in the current code. The method Dr. van de Sande decided to switch to involved using constraint states, which are not actual measurable quantities. Their usefulness, however, comes in the simplification of the actual construction of the Hamiltonian. The complication arises in the form of a generalized eigenvalue problem, which must be dealt with. This problem is purely a mathematical problem, separate from the physics, and became the focus of my summer's work.

The mathematical problem is:

$$\mathbf{M}\vec{b} = \lambda\mathbf{E}\vec{b} \tag{1}$$

where $\mathbf{M}$ is Hermitian and $\mathbf{E}$ is positive, diagonal, semi-definite. This is solved by breaking $\mathbf{M}$ and $\mathbf{E}$ into blocks of the form:

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\dagger & \mathbf{C} \end{pmatrix} \qquad \mathbf{E} = \begin{pmatrix} \mathbf{E}' & 0 \\ 0 & 0 \end{pmatrix} \tag{2}$$

where $\mathbf{E}'$ and $\vec{b}$ are of the form:

$$\mathbf{E}' = \begin{pmatrix} e_1 & 0 & 0 & \\ 0 & e_2 & 0 & \cdots \\ 0 & 0 & e_3 & \\ & \vdots & & \ddots \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} \vec{x} \\ \vec{y} \end{pmatrix}. \tag{3}$$

Thus $\vec{x}$ contains the normal states and $\vec{y}$ contains the constraint states. After doing some algebra, the problem is reduced to the form

$$(\mathbf{A} - \mathbf{B}\mathbf{C}\mathbf{B}^\dagger)\vec{x} = \lambda\mathbf{E}'\vec{x}. \tag{4}$$

This form can be further reduced to a more typical eigenvalue problem form by letting $\vec{x_i'} = \sqrt{e_i}\vec{x_i}$ and $\mathbf{J} = \mathbf{A} - \mathbf{BCB}^\dagger$, thus giving a final form of:

$$\mathbf{J}\vec{x'} = \lambda\vec{x'} \tag{5}$$

which is that of the traditional eigenvalue problem, which is easily solved.

The first thing I had to do with this new method was to change the old Mathematica code that Jonathon Bratt and Beth Watson had written several summers earlier, and Joe Berg and I had added to last summer. First, I had to write a new routine for creating the basis that included the constrained states. The old notation was that a $b^\dagger$ represented a quark, and a $d^\dagger$ represented an anti-quark. Dr. van de Sande decided to use capital letters to represent the constrained quarks, $B^\dagger$, and anti-quarks, $D^\dagger$. Now the basis could contain any combination of $b^\dagger$'s, $d^\dagger$'s, $B^\dagger$'s, and $D^\dagger$'s. Next, I had to edit out a good chunk of the code to construct the Hamiltonian, leaving very little of what was originally written. All of the code Joe Berg and I had written the previous summer for the chiral four-fermion interaction was salvaged. However, along with the rest of the remaining code, it needed to be adjusted to handle the constraint states properly. Once this was done, results from the new code were compared with results from the old code to check it's accuracy. While checking, an issue arose with the κ_5 term, where the coupling constant in the new old method needed to be adjusted to match correctly. Dr. van de Sande was able to determine the relationship between the two methods to be:

$$\kappa_5' = \frac{\kappa_5}{1 + \kappa_5 f(k)} \tag{6}$$

where κ_5 is used in the old method and κ_5' in the new method. Dr. van de Sande figured that $f(k)$ was a complicated logarithmic function, so I ran many numerical tests in order to try and estimate it's value for different values of k. The tests were able to determine that it's form appeared to be:

$$f(k) = -\frac{2}{\pi}\sum_{l=\frac{1}{2}}^{k-\frac{1}{2}}\frac{1}{l} \tag{7}$$

Using this formula for various values of momentum, k, the old and new methods agree within expected machine error.

After we got all of the new code to match all of the old code, we hit a snag which forced me to change gears. I was now asked to teach myself FORTRAN77, so that I could write a subroutine that Dr. van de Sande could call in his C code to solve this eigenvalue problem. The only books I could find to help me learn FORTRAN77 were a VAX FORTRAN and a FORTRAN90 book. These two books were not nearly as helpful as I'd have liked, but I was still able to learn enough from them and Dr. van de Sande to accomplish my task. First, I wrote a subroutine which solved the eigenvalue problem for dense matrices; then I had to write a subroutine for sparse matrices. Sparse matrices contain mostly zero entries and therefore are stored in different formats in order to cut down on memory usage and allow for extremely large matrices. The dense routine I was able to write using routines from the BLAS and LAPACK standard libraries. Dr. van de Sande then linked to my code and began using it in his code.

The sparse routine was more complicated, since BLAS and LAPACK do not have very high level routines for sparse matrices. I had to learn how to use the routine ZLANDR3

to perform the Lanczos's method to solve for the eigenvalues and eigenvectors. The other major difficulty was finding a routine, or method, to solve the equation $\mathbf{C}\vec{q} = \vec{x}$ for $\vec{q}$, where $\mathbf{C}$ is sparse and Hermitian. After searching the internet for a day or two, we decided to use a routine called ME27 from the HSL library to solve for $\vec{q}$. Other than one small problem, this routine did what we needed. The problem had to do with memory allocation and the variable that was supposed to report the amount of memory needed did not report a large enough value. This coupled with a memory allocation problem built into FORTRAN77, the COMMON block, caused us to look to the language C to complete the project.

The sparse code in FORTRAN77 worked, but the memory problems were beyond my ability to fix. My next task became to learn C and reconstruct my FORTRAN code into C code. Also, I needed to find a new routine to replace ME27, due to its built in problem. After searching the internet once again, we found a presentation by a woman at a conference documenting different capabilities of several sparse linear solvers. Using this list we were able to narrow down the possibilities to one, SPOOLES. SPOOLES appeared to be the only sparse linear solver which could take advantage of the Hermitian structure of the matrix we needed solved. After several days of reading and learning about how to use SPOOLES, I eventually got it working. However, I did have to make a change to the Bridge library in the LinSol directory. There was an error such that it did not handle Hermitian matrices correctly. It was a small error,easily fixed and documented, in the SPOOLES library in my directory. After lots of work and testing, it appeared to work for any matrix up to a certain size. With the help of Dr. van de Sande, the error was found to be in SPOOLES once again. This time they had used the data type int, and the number we were using was larger than that data type could handle; so it was changed to long int and documented once again.

In the midst of writing the C code, I had to take a break to add a flag and calculation to the dense code I had written prior to the sparse code. This consisted of allowing the user to request and recieve the full eigenvectors, which included the constraint states. Dr. van de Sande had initially not needed these, but had discovered an operator which required this knowledge. I also added this to the sparse code once I had finished getting the dense code to run properly. The next step for the sparse code was then to make it run faster and more efficiently, using as little memory as possible. I was able to speed it up considerably, and suprisingly the rate at which the time increases is a smaller exponent of the momentum, k, than the old method (however for small matrices it takes longer due to a large coefficient). The next step was to try and save memory and still speed it up further. Dr. van de Sande realized that the matrix $\mathbf{C}$ was actually real and not imaginary. Using this knowlege, I changed the code to store and treat $\mathbf{C}$ as a symmetric, real matrix. This should have reduced the memory usage for $\mathbf{C}$ by a factor of 2, which would reduce the overall memory usage. Dr. van de Sande also expected this to speed up the code some amount; however, after running some preliminary tests this way of handling $\mathbf{C}$ actually slowed down the code. This result baffled Dr. van de Sande.

The last week of work was rather frustrating. For while I was trying to run tests to find out why it was slowing down when I treated $\mathbf{C}$ as a real, symmetric matrix, the computers went down. A hard drive decided to break, and all kinds of difficulties in installing a new operating system arose. I was eventually able to find out that SPOOLES once again is to blame for the slow down. The solving step is too long even for real matrices, so doing

it twice takes up more time than it saves. The best way to fix this would be to have $\mathbf{C}$ be a real matrix and the vector be complex, but again SPOOLES does not know how to handle this situation properly. So instead, I had to put the real and complex parts as two vectors and do the solve with two right hand sides. This saved only a minimal amount of memory and time, but not enough to be terribly useful. However, this is the correct way to handle the situation. The compiler flag was left in, but turned off to make the code more general, since the time and memory saved was minimal. Since SPOOLES seems to be the deficiency in the program, another method should be used. Due to time running out on this summer, this will have to be fixed at a later date by another person.