Code Structure: Twobody Nuclear Reaction Calculations

December 30, 2025

1 Overview

The twobody code computes nuclear reaction amplitudes by convoluting process-specific kernels with 2N density matrices. The code is split into two independent components that communicate through well-defined interfaces.

1.1 Two-Component Architecture

Mantle Code (`varsub-twobodyvia2Ndensity/`): Process-independent framework that handles quantum number summations, angular integrations, and density matrix interpolation. Works in units of **fm**.

Kernel Code : Process-specific physics implementation. Works in units of **MeV**. Two implementations are currently available:

- `varsub-PionPhotoProdThresh.twobody/`: Pion photoproduction at threshold
- `varsub-PionPion.twobody/`: Pion-pion scattering

1.2 Unit Conversion

The mantle code (fm) and kernel code (MeV) use different unit systems. Unit conversion occurs in `finalstatesums.twobodyvia2Ndensity.f` via the factor $(HC)^3/(2\pi)^3$. If the kernel has units MeV^{-n}, the final output has units MeV^{3-n}.

2 Call Hierarchy

2.1 Main Control Flow

Level 1: `varsub-main.twobodyvia2Ndensity.f`
Main program: controls energy/angle loops, reads input/density files, outputs results

Level 2: `varsub-finalstatesums.twobodyvia2Ndensity.f`
Function: `twobodyfinalstatesumsvia2Ndensity()`
Loops over final state quantum numbers, accumulates contributions

Level 3: `varsub-calculate2BI2.f`
Function: `Calculate2BIntegralI2()`
Performs angular integrals and spin sums, interpolates density matrix

Level 4: `varsub-2Bkernel.PionPhotoProdThresh.f`
Function: `Calc2Bspinisospintrans()`
Computes process-specific kernel amplitudes

Level 5: `varsub-2Bspinsym.PionPhotoProdThresh.f` (and `*asym.f`)
Functions: `CalcKernel2B*()` and `StaticKernel*()`
Evaluate spin structures for specific diagrams

3 File Descriptions

3.1 Mantle Code Files

3.1.1 `varsub-main.twobodyvia2Ndensity.f`

Purpose: Program entry point and orchestration
Responsibilities:

- Parse input file (energies, angles, nucleus properties, quadrature parameters)
- Set up radial and angular quadrature grids
- Loop over energies and scattering angles
- Read 2N density matrix from HDF5 files
- Loop over initial quantum numbers (mt12, j12, s12, l12, m12, ip12)
- Call `twobodyfinalstatesumsvia2Ndensity()` for each configuration
- Write results to output file

Key Variables:

- `Result(extQnum, twoMzp, twoMz)`: Output array storing computed amplitudes
- `P12MAG(ip12)`: Radial momentum grid points
- `AP12MAG(ip12)`: Radial momentum integration weights
- `rhoDensity`: 2N density matrix (read from file, stored in module)

3.1.2 `varsub-finalstatesums.twobodyvia2Ndensity.f`

Purpose: Sum over final state quantum numbers
Responsibilities:

- Loop over final state quantum numbers: j12p, s12p, l12p, m12p, ip12p, twoMzp, twoMz
- Call `Calculate2BIntegralI2()` to compute angular integrals
- Multiply by integration weights (momentum weights, phase space factors)
- Apply unit conversion factor: $(HC)^3/(2\pi)^3$
- Accumulate into `Result()` array

Key Constraint: mt12p = mt12 (isospin projection is conserved)

3.1.3 `varsub-calculate2BI2.f`

Purpose: Perform angular integrals and spin sums

Responsibilities:

- Loop over spin projections (ms, msp) of the (12) subsystem
- Nested loops over angular quadrature points: (θ_{12}, ϕ_{12}) and (θ'_{12}, ϕ'_{12})
- Convert spherical to Cartesian coordinates
- Compute spherical harmonics
- Call `Calc2Bspinisospintrans()` to get kernel amplitudes
- Interpolate density matrix at required momentum values (bilinear interpolation)
- Weight by spherical harmonics, quadrature weights, and Clebsch-Gordan coefficients
- Sum over all angular points and spin projections

Quadrature Options:

- Gaussian: Separate 1D quadratures for θ and ϕ
- Lebedev-Laikov: Spherical quadrature on unit sphere

3.1.4 Supporting Files in `varsub-twobodyvia2Ndensity/`

- `varsub-read2Ndensity.f`: Reads HDF5 density files into module variables
- `varsub-setquads.f`: Sets up angular quadrature grids
- `varsub-LebedevLaikov.f`: Lebedev-Laikov quadrature implementation
- `varsub-spinstructures.f`: General spin algebra routines (`singlesigmasym()`, `doublesigmasym()`)

3.2 Kernel Code Files

3.2.1 `varsub-2Bkernel.PionPhotoProdThresh.f`

Purpose: Main kernel computation for pion photoproduction

Responsibilities:

- `KernelGreeting()`: Print process name and version to stdout
- `KernelFarewell()`: Print description of computed quantities and units
- `Calc2Bspinisospintrans()`: Main kernel calculation routine
  - Receives momentum vectors (pVec, uVec) and quantum numbers
  - Performs variable substitution to simplify momentum transfers
  - Computes momentum vectors for use in diagrams (qVec, qpVec, kVec, kpVec)
  - Calls diagram-specific routines based on `calctype` parameter
  - Returns `Kernel2B(diagNum, extQnum, s12p, msp, s12, ms)`

- `getDiagAB()`: Computes leading order contributions
- `getStaticDiags()`: Computes next-to-leading order corrections

Chiral Order Organization:

- O(δ^2): Leading order contributions
- O(δ^4): Next-to-leading order corrections

Calculation terminates at the order specified by `calctype`.

Variable Substitution:

The code uses $\vec{u} = \vec{p}_{12} - \vec{p}'_{12} + \vec{k}/2$ as the integration variable instead of $\vec{p}'_{12}$. This simplifies the momentum transfer and introduces a Jacobian factor of -1.

3.2.2 `varsub-2Bspinsym.PionPhotoProdThresh.f`

Purpose: Spin structures for symmetric diagrams (s12p = s12)

Functions:

- `CalcKernel2BAsym()`: Leading order contribution (type A)
- `CalcKernel2BBsymVec()`: Leading order contribution (type B)
- `StaticKernelAsym()`: NLO correction (type A)
- `StaticKernelBsym()`: NLO correction (type B)
- `StaticKernelCsym()`: NLO correction (type C)
- `StaticKernelDsym()`: NLO correction (type D)
- `StaticKernelEsym()`: NLO correction (type E)

Each function receives prefactors and momentum vectors, computes spin matrix elements, and adds contributions to the kernel array.

3.2.3 `varsub-2Bspinasy.PionPhotoProdThresh.f`

Purpose: Spin structures for antisymmetric diagrams (s12p $\neq$ s12)

Contains analogous functions to the symmetric file but for spin-flip transitions: `CalcKernel2BAasy()`, `CalcKernel2BBasyVec()`, `StaticKernelAasym()`, etc.

3.2.4 Other Kernel Files

- `varsub-calculateQs.PionPhotoProdThresh.f`: Momentum kinematics calculations
- `varsub-usesymmetries.PionPhotoProdThresh.f`: Symmetry relations (currently not used)
- `readinput.twobody.PionPhotoProdThresh.f`: Kernel-specific input parsing

3.3 Pion-Pion Scattering Kernel Files

The pion-pion scattering kernel (`varsub-PionPion.twobody/`) implements elastic pion scattering from nuclei based on chiral perturbation theory. The structure closely parallels the photoproduction kernel but with different physics and kinematics.

3.3.1 `varsub-2Bkernel.PionPion.f`

Purpose: Main kernel computation for pion-pion scattering

Responsibilities:

- `KernelGreeting()`: Print process name, compute probe energy from gamma energy
  - Converts lab-frame energy to probe pion energy via relativistic kinematics
  - Handles near-threshold kinematics where k^2 may be slightly negative
- `KernelFarewell()`: Print description (references BKM review equation 5.30)
- `Calc2Bspinisospintrans()`: Main kernel calculation routine
  - Performs variable substitution: $\vec{u} = \vec{p}_{12} - \vec{p}'_{12} + (\vec{k} + \vec{k}')/2$
  - Calculates final pion momentum $\vec{k}'$ using `calculateqs2Mass()`
  - Computes momentum transfer vector $\vec{q}$
  - Applies overall prefactor: $8\pi\sqrt{s}$ where $\sqrt{s} = E_{\text{nuc}} + E_\pi$
  - Calls `getDiagAB()` to compute kernel contributions
  - Returns kernel in units MeV^{-3} (so Result has units MeV^0)
- `getDiagAB()`: Computes diagram contributions (A, B, C)
  - Diagram A: Contact interaction, prefactor $\propto 1/q^2$
  - Diagram B: Single-sigma structure, prefactor $\propto 1/(q^2 + m_\pi^2)$
  - Diagram C: Double-sigma structure, prefactor $\propto 1/(q^2 + m_\pi^2)^2$
  - All diagrams include isospin factors depending on (t_{12}, m_{t12})
  - Separates symmetric (s12p = s12) and antisymmetric (s12p $\neq$ s12) contributions

Physics Basis:

- Based on BKM (Bernard-Kaiser-Meißner) review equation 5.30
- Implements chiral EFT for pion-nucleon interactions
- Reduced mass: $\mu = m_\pi/m_N$
- Base prefactor: $\frac{1}{32(1+\mu)(\pi f_\pi)^4}$ where $f_\pi = 92.42$ MeV

Kinematics:

- Elastic scattering: $\pi + N \to \pi + N$
- Uses center-of-mass kinematics via `calculateqs2Mass()`
- Momentum vectors: $\vec{p}, \vec{p}', \vec{k}, \vec{k}'$ for initial/final nucleon and pion momenta
- Momentum transfer: $\vec{q} = \vec{p} - \vec{p}' + (\vec{k} + \vec{k}')/2$

3.3.2 `varsub-2Bspinsym.PionPion.f`

Purpose: Spin structures for symmetric diagrams (s12p = s12)
Functions:

- `CalcKernel2BAsym()`: Diagram A contribution
  - Pure isospin structure, no momentum dependence
  - Includes factor $(-1)^{t_{12}}\delta_{m_{t12},0}$
  - Diagonal in spin: $\delta_{s'_{12},s_{12}}\delta_{m'_s,m_s}$
- `CalcKernel2BBsym()`: Diagram B contribution
  - Calls `doublesigmasym()` for $\vec{\sigma}_1 \cdot \vec{q}\,\vec{\sigma}_2 \cdot \vec{q}$ structure
  - Isospin factor: $(2t_{12}(t_{12}+1) - 3)$
- `CalcKernel2BCsym()`: Diagram C contribution
  - Same spin structure as B but different momentum dependence
  - Includes $(q^2 + m_\pi^2)^{-2}$ instead of $(q^2 + m_\pi^2)^{-1}$
- `CalcKernel2BDsym()`: Diagram D (not currently used)

3.3.3 `varsub-2Bspinasy.PionPion.f`

Purpose: Spin structures for antisymmetric diagrams (s12p $\neq$ s12)
Contains analogous functions for spin-flip transitions:

- `CalcKernel2BAasy()`: Spin-flip version of diagram A
- `CalcKernel2BBasy()`: Uses `doublesigmaasy()` instead of `doublesigmasym()`
- `CalcKernel2BCasy()`: Spin-flip version of diagram C
- `CalcKernel2BDasy()`: Diagram D (not implemented)

3.3.4 `varsub-calculateQs.PionPion.f`

Purpose: Momentum kinematics for pion-pion scattering
Key Subroutines:

- `calculateqs2Mass()`: Elastic scattering kinematics
  - Input: momenta $\vec{p}, \vec{p}', \vec{k}$ and masses m_1, m_2, m_3, m_4
  - Calculates final pion momentum $\vec{k}'$ from energy-momentum conservation
  - Uses Mandelstam $s = (E_1 + E_2)^2$ in CM frame
  - Output momentum magnitude: $|\vec{k}'| = \sqrt{E_4^2 - m_4^2}$
  - Direction: $\vec{k}' = (0, |\vec{k}'|\sin\theta_{\rm cm}, |\vec{k}'|\cos\theta_{\rm cm})$
- `CalculateQs()`: Legacy routine (not currently used in PionPion)

Kinematic Relations:

- $E_1 = \sqrt{m_1^2 + \vec{k}^2}$, $E_2 = \sqrt{m_2^2 + \vec{k}^2}$
- $\sqrt{s} = E_1 + E_2$
- $E_3 = \frac{1}{2\sqrt{s}}(s + m_3^2 - m_4^2)$
- $E_4 = \sqrt{s} - E_3$

3.3.5 Other PionPion Kernel Files

- `varsub-usesymmetries.PionPion.f`: Symmetry relations (currently not used)
- `varsub-readinput.twobody.PionPion.f`: Kernel-specific input parsing

3.4 Comparison: Photoproduction vs Pion-Pion

Feature	**Photoproduction**	**Pion-Pion**
Process	$\gamma + N \to \pi + N$	$\pi + N \to \pi + N$
Theory	Threshold expansion	Chiral EFT (BKM 5.30)
Orders	O(δ^2), O(δ^4)	Single order
Diagrams	A, B, Static (A–E)	A, B, C
Kinematics	Photoproduction threshold	Elastic scattering
Output units	fm^{-1}	MeV^0 (dimensionless)
Kernel units	MeV^{-2}	MeV^{-3}

Common Features:

- Both use variable substitution in momentum integration
- Both separate symmetric/antisymmetric spin contributions
- Both use same mantle code infrastructure
- Both implement isospin algebra
- Both call `doublesigmasym()/doublesigmaasy()` from `varsub-spinstructures.f`

4 Data Flow

4.1 Input Data

1. **Input file** (text): Energies, angles, quadrature parameters, file paths
2. **Density file** (HDF5): 2N density matrix $\rho(p_{12}, p'_{12}, \text{quantum numbers})$

4.2 Computational Flow

```
Main Program
    | (loops: energy, angle, quantum numbers)
    |
    +---> Read Density Matrix (once per energy/angle)
    |
```

```
    +---> twobodyfinalstatesumsvia2Ndensity()
            | (loop: final state quantum numbers)
            |
            +---> Calculate2BIntegralI2()
                    | (loops: spin projections, angles)
                    | (interpolate density)
                    |
                    +---> Calc2Bspinisospintrans()
                            | (compute momenta)
                            |
                            +---> getDiagAB()
                            |       +---> CalcKernel2B*sym/asy()
                            |
                            +---> getStaticDiags()
                                    +---> StaticKernel*sym/asy()
                                    +---> (multiple contributions)
                    |
                    | <--- returns: Kernel2B(...)
                    |
                | (weight and sum)
            |
            | <--- returns: Int2B(diagNum, extQnum)
            |
        | (accumulate with weights)
    |
    | <--- updates: Result(extQnum, twoMzp, twoMz)
    |
+---> Write Results to File
```

4.3 Output Data

1. **Output file** (text): `Result(extQnum, twoMzp, twoMz)` for each energy and angle
2. **extQnum**: Index for external quantum numbers
   - For photoproduction: polarization indices (1=x, 2=y, 3=z)
   - For pion-pion: typically 1 (no polarization dependence)
3. **twoMzp, twoMz**: Initial and final nuclear spin projections (times 2)

Output units vary by process:

- Pion photoproduction: fm^{-1} (proportional to F_{TL} functions)
- Pion-pion scattering: MeV^0 (dimensionless, proportional to scattering amplitude)

5 Key Interfaces

5.1 Mantle → Kernel Interface

The mantle code calls `Calc2Bspinisospintrans()` with:

Inputs:

- `pVec(3)`: Physical momentum vector in MeV
- `uVec(3)`: Integration variable vector in MeV
- Quantum numbers: `ml12, ml12p, t12, mt12, t12p, mt12p, l12, s12, l12p, s12p`
- Kinematics: `thetacm, Eprobe, Mnucl`
- Control: `calctype, numDiagrams, extQnumlimit`

Outputs:

- `Kernel2B(diagNum, extQnum, s12p, msp, s12, ms)`: Complex kernel amplitudes
- `ppVecs(diagNum, 1:3)`: Transformed momentum vectors for each diagram

The kernel has no knowledge of:

- Density matrices
- Angular integrals or quadratures
- Final state summations
- Output file formats

5.2 Density Matrix Interface

The density matrix is read once per energy/angle and stored in a Fortran module (`CompDens`). It is accessed throughout the calculation via:

- `rhoDensity(ip12, ip12p, rindx)`: 3D array
- `ip12, ip12p`: Momentum grid indices
- `rindx`: Composite index for all quantum numbers of the (12) channel

The mantle code performs bilinear interpolation to evaluate the density at arbitrary momentum values needed for the kernel.

6 Extensibility

6.1 Adding a New Process

To compute a different reaction (e.g., Compton scattering, kaon production), follow these steps. The pion photoproduction and pion-pion scattering kernels serve as reference implementations.

1. Create new kernel directory: `varsub-<Process>.twobody/`
2. Implement required subroutines:
    - `KernelGreeting()`: Identify your process, compute kinematic variables
    - `KernelFarewell()`: Describe your output and units

- `Calc2Bspinisospintrans()`: Main kernel computation
  - Set up kinematics and momentum vectors
  - Call diagram calculation routines
  - Apply overall prefactors
  - Return kernel with proper units
- Diagram-specific spin structure routines (sym/asy versions)
- Kinematic calculation routines (analogous to `calculateqs2Mass()`)

3. Ensure proper unit handling:
   - Kernel should work in MeV
   - Choose units so Result() has desired output units
   - Remember: Result units = MeV^{3-n} if kernel has units MeV^{-n}
4. The mantle code remains completely unchanged
5. Update Makefile to link new kernel files

Key Design Principles:

- Keep kernel code independent of mantle infrastructure
- Separate symmetric and antisymmetric spin contributions
- Use variable substitution if it simplifies integrals (include Jacobian!)
- Implement isospin algebra explicitly in diagram prefactors
- Document physics basis (review paper, equation numbers)

6.2 Adding Higher Orders

To add higher order contributions (e.g., $O(\delta^6)$):

1. Add new diagram calculation routines in kernel files
2. Update `Calc2Bspinisospintrans()` to call new diagrams when appropriate `calctype` is set
3. Update `calctype.def` to include new order definitions
4. No changes needed to mantle code structure

7 Important Conventions

7.1 Quantum Numbers

- Nuclear spin quantum numbers are stored as `twoMz` $= 2M_z$ (integers)
- Allows half-integer spins to be represented exactly
- (12) subsystem quantum numbers (j12, s12, l12, etc.) are integers (0 or 1 for spin)
- All loops over `twoMz` use steps of 2: `do twoMz = twoSnucl, -twoSnucl, -2`

7.2 Array Indexing

- `Result(extQnum, twoMzp, twoMz)`
  - extQnum: 1 to extQnumlimit (typically 1–3 for polarizations)
  - twoMzp, twoMz: -twoSnucl to +twoSnucl in steps of 2
- `Kernel2B(diagNum, extQnum, s12p, msp, s12, ms)`
  - diagNum: 1 to numDiagrams (currently 1 for this process)
  - s12p, s12: 0 or 1
  - msp, ms: -1, 0, or +1 (but only -s12 to +s12 are physical)

7.3 Module Usage

The code uses a Fortran module `CompDens` (defined in `CompDens.F90`) to share density matrix data without passing large arrays through function arguments. This module is imported via `USE CompDens` in relevant files.

8 Summary

The code architecture cleanly separates process-independent framework (mantle) from process-specific physics (kernel):

Mantle Handles all bookkeeping: quantum number loops, angular integrals, density interpolation, unit conversion, I/O. Works in fm.

Kernel Implements physics: Feynman diagrams, spin structures, momentum kinematics. Works in MeV. Two kernels currently implemented:

- **Pion Photoproduction**: Threshold expansion with $O(\delta^2)$ and $O(\delta^4)$ contributions
- **Pion-Pion Scattering**: Chiral EFT elastic scattering (BKM review)

Interface Well-defined: kernel receives momenta and quantum numbers, returns amplitudes. No shared knowledge of implementation details.

Result Reusable framework. Same mantle code used for multiple processes by swapping kernel implementation. This document demonstrates extensibility with two working examples.

Code Organization:

- Mantle files: `varsub-twobodyvia2Ndensity/*`
- Photoproduction kernel: `varsub-PionPhotoProdThresh.twobody/*`
- Pion-pion kernel: `varsub-PionPion.twobody/*`
- Common utilities: `common-densities/*`