VecAmpFit: библиотека для амплитудных анализов с векторизованными вычислениями

Кирилл Чиликин

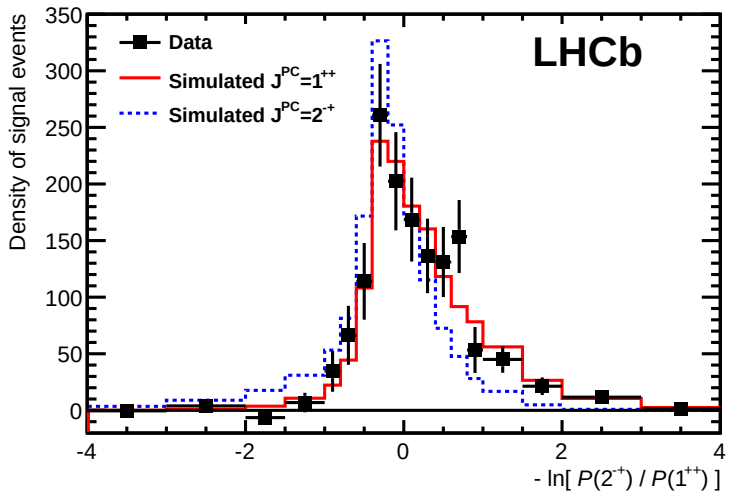
Институт ядерной физики имени Г.И. Будкера СО РАН, Новосибирск

Сессия-конференция секции ядерной физики ОФН РАН
21 февраля 2025

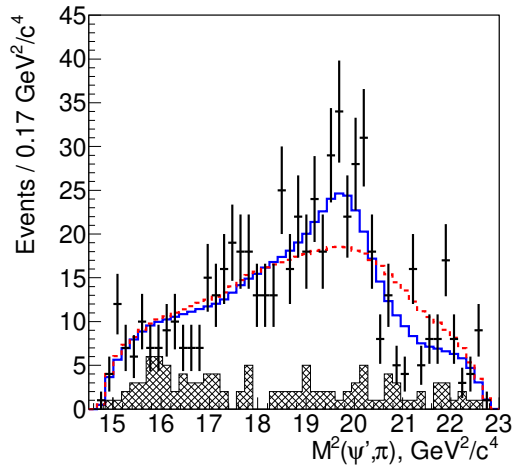
Амплитудные анализы в спектроскопии кваркония

Амплитудный анализ - подгонка экспериментальных данных зависимостью функции плотности сигнала от фазового пространства. Многомерный амплитудный анализ - наиболее чувствительный метод для анализа сложных цепочек распадов, поскольку позволяет использовать всю доступную информацию о зависимости плотности сигнала от фазового пространства. Такие анализы дают много интересных физических результатов. Некоторые примеры из спектроскопии кваркония, которая мотивировала данную работу, включают измерение квантовых чисел $\chi_{c1}(3872)$ [LHCb: PRL 110, 222001 (2013), PRD 92, 011102 (2015)], исследования $T_{c\bar{c}1}(4430)^+$ [Belle: PRD 88, 074026 (2013); LHCb: PRL 112, 222002 (2014)], обнаружение множества экзотических состояний в распаде $B^+ \to J/\psi\phi K^+$ [LHCb: PRL 118, 022003 (2017), PRL 127, 082001 (2021)], измерение квантовых чисел $T_{b\bar{b}1}(10610)^+$ и $T_{b\bar{b}1}(10650)^+$ [Belle: PRD 91, 072003 (2015)] и другие.

LHCb $\chi_{c1}(3872)$ PRL 110, 222001 (2013)



Belle $T_{c\bar{c}1}(4430)^+$ PRD 88, 074026 (2013)



Существующие библиотеки

Вычислительная сложность амплитудных анализов и использование схожих амплитуд приводят к необходимости создания специализированных библиотек.
Императивный подход (явное программирование всех вычислений):

- AmpGen: https://github.com/GooFit/AmpGen
  Подгонка многочастичных распадов моделью изобар.
- AmpTools: https://github.com/mashephe/AmpTools
  Небинированная подгонка экспериментальных данных методом максимума правдоподобия когерентной суммой амплитуд.
- Laura^{++}: https://laura.hepforge.org/
  Publication: Comput. Phys. Commun. **231**, 198.
  Библиотека для Далиц-анализов.

Декларативный подход (амплитуда задаётся как функция):

- Проект ComPWA: https://compwa.github.io
  QRules: генерация разрешённых переходов между состояниями на основе законов сохранения квантовых чисел; AmpForm: формулировка моделей амплитуд для различных спиновых формализмов и с разной динамикой; TensorWaves: пакет для подгонки и генерации данных для различных вычислительных бэкендов.
- TensorFlowAnalysis2: https://github.com/apoluekt/TFA2
  Анализ данных с использованием TensorFlow 2.
- TF-PWA: https://github.com/jiangyi15/tf-pwa
  Парциально-волновой анализ с использованием TensorFlow.

VecAmpFit

VecAmpFit - новая библиотека, которая была создана для выполняемого в настоящее время амплитудного анализа в эксперименте Belle II. Основное предназначение библиотеки - работа над многомерными анализами, поэтому она не ограничена какой-либо конкретной моделью, а включает код для подгонки и функции, необходимые для вычисления амплитуд. Функции плотности сигнала и фона должны быть запрограммированы пользователем. Императивный подход, предпринимаются меры для обхода его ограничений в оптимизации, язык - современный стандарт Fortran 2018.

Метод оптимизации или параллельных вычислений	Поддержка в VecAmpFit
Аппаратный параллелизм	Все вычисления векторизованы
Исключение условных выражений	Конфигурация через препроцессор
Градиент	Нужно явное программирование
Гессиан	Не поддерживается
Параллельные нити	С использованием OpenMP
Параллельные процессы	OpenMPI не поддерживается
Выгрузка на GPU	С использованием OpenACC

$D^0 \to K^-\pi^+\pi^0$

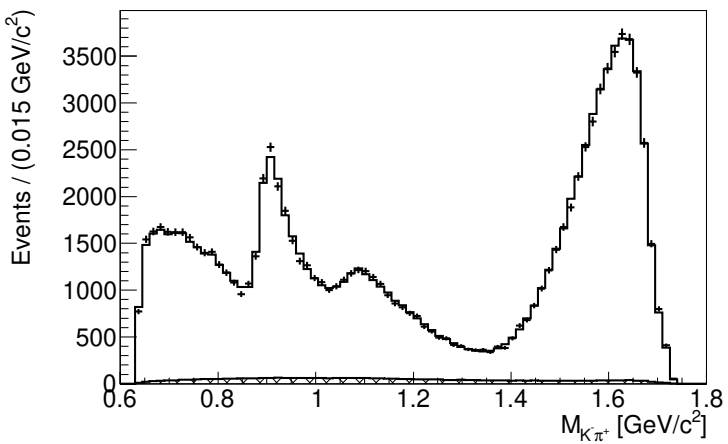
Пример основан на анализе CLEO [PRD **63**, 092001 (2001)]. Амплитуда равна

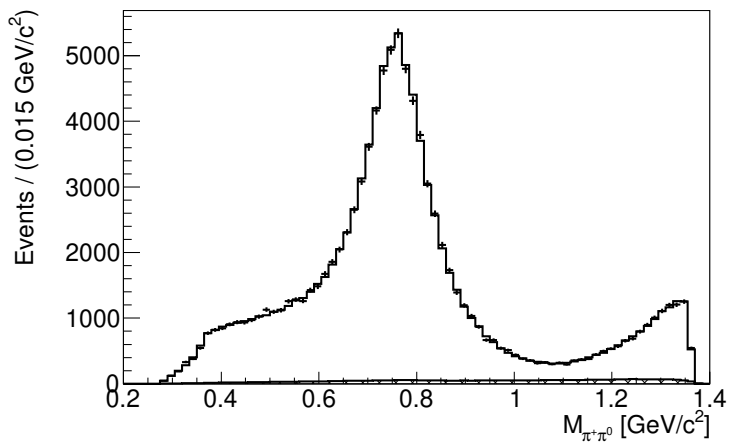
$$A(\Phi) = a_{nr} e^{i\phi_{nr}} + \sum_R a_R e^{i\phi_R} \frac{F_D^{(\mathrm{BW})}(q, r^{(D)}, J)}{F_D^{(\mathrm{BW})}(q_0, r^{(D)}, J)} B_R^{(\text{no } q)}(M_{AB}) \mathcal{A}_J(ABC|R) \qquad (1)$$

где $F_D^{(BW)}$ - формфактор Блатта-Вайсскопфа, q и q_0 - импульсы D для инвариантной массы комбинацмм AB и массы резонанса R, соответственно, B - амплитуда Брейта-Вигнера, угловая часть равна

$$\mathcal{A}_0(ABC|R) = 1;\ \mathcal{A}_1(ABC|R) = M_{AC}^2 - M_{BC}^2 + \frac{(M_D^2 - M_C^2)(M_B^2 - M_A^2)}{M_R^2}. \qquad (2)$$

Модель включает 7 резонансов: $\rho(770)^+$ и $\rho(1700)^+$ в системе $\pi^+\pi^0$; $K^*(892)^-$, $K_0^*(1430)^-$ и $K^*(1680)^-$ в системе $K^-\pi^0$; $\bar{K}^*(892)^0$ и $\bar{K}_0^*(1430)^0$ в системе $K^-\pi^+$.



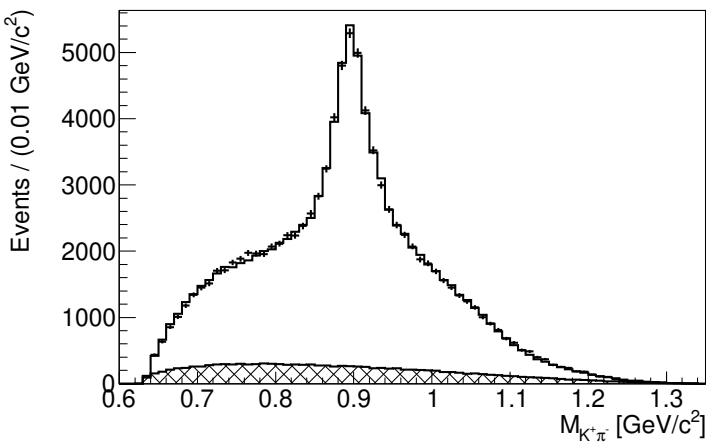


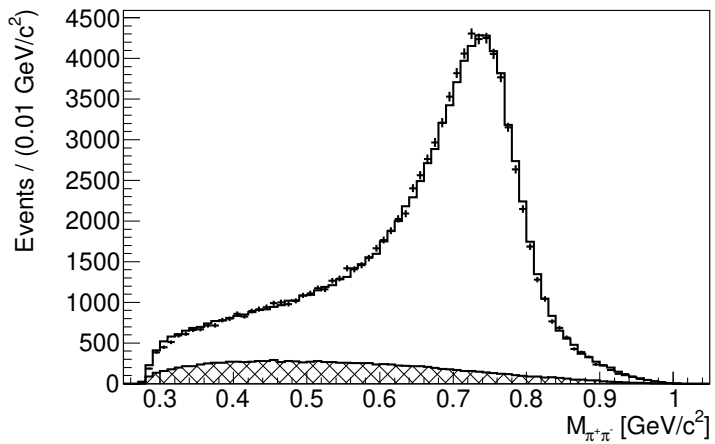
$e^+e^- \to K^+K^-\pi^+\pi^-$

Анализ CMD-3 [PLB **756**, 153 (2016)] (не полный амплитудный анализ, были ограничения на фазы): наибольший вклад - $e^+e^- \to K_1(1270)^+K^-$, $e^+e^- \to K_1(1400)^+K^-$ - только эти вклады реализованы в примере. Амплитуда для цепочки распадов $\gamma^* \to K_J^{(1)}\bar{K}^{(2)}$, $K_J^{(1)} \to K^{(1)}\pi^{(1)}\pi^{(2)}$ равна

$$A^{\gamma^*\to K_J^{(1)}\bar{K}^{(2)}}_{\lambda_{\gamma^*}}(\Phi) = \sum_{\lambda_{K_J^{(1)}}=-1,0,1} d^1_{\lambda_{\gamma^*}\,\lambda_{K_J^{(1)}}}(\theta^{(1)}_{\gamma^*}) \sum_{K_J^{(1)}} H^{(\gamma^*\to K_J^{(1)}\bar{K}^{(2)})}_{\lambda_{K_J^{(1)}}\,0} B_{K_J^{(1)}}(M_{K^{(1)}\pi^{(1)}\pi^{(2)}})$$
$$\times \left(A^{(K_J^{(1)}\to K_J^{*(11)}\pi^{(2)})}_{\lambda_{K_J^{(1)}}}(\Phi) + A^{(K_J^{(1)}\to K_J^{*(12)}\pi^{(1)})}_{\lambda_{K_J^{(1)}}}(\Phi) + A^{(K_J^{(1)}\to \rho_J K^{(1)})}_{\lambda_{K_J^{(1)}}}(\Phi) \right).$$

Одновременно подгоняется 5 точек по энергии: 1800, 1850, 1900, 1950 и 2000 МэВ



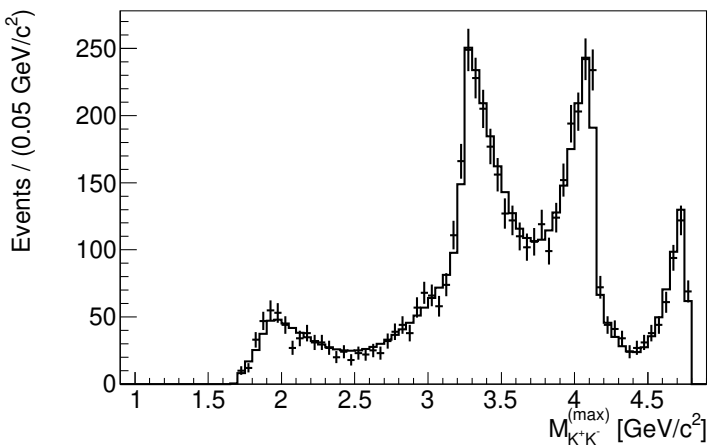


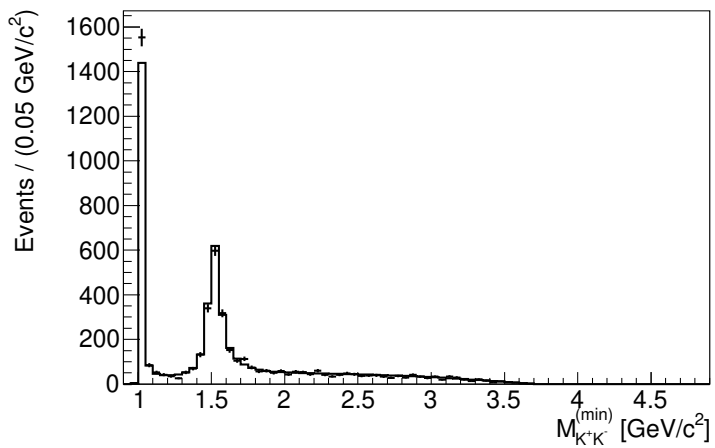
$B^+ \to K^+K^+K^-$

Модель соответствует модели `GenFit3K` из LAURA^{++} за исключением нормировок амплитуд отдельных резонансов и предназначена для сравнения производительности.

$$
\begin{aligned}
A_R^{(\text{ang})}(\Phi) &= (p^* q)^L F_L^{(\text{cov})}(m_{12}) P_L^{(\text{Leg LAURA}^{++})}(\cos\theta_R), \\
A_{\phi_J}(\Phi) &= B_{\phi_J}^{(\text{no } q)}(m) A_{\phi_J}^{(\text{ang})}(\Phi) F_{B^+}^{(BW)}(q(F_J^{(\text{cov})}(m_{12}))^{\frac{1}{J}}, J, r^{(B)}), \\
A(\Phi) &= a_0 + \sum_R a_R \left[A_R^{(B^+ \to \phi_J K^+_{(1)})}(\Phi) + A_R^{(B^+ \to \phi_J K^+_{(2)})}(\Phi) \right].
\end{aligned}
\tag{3}
$$

Для распада $0 \to R + 3$, $R \to 1 + 2$ p - импульс 3 в системе покоя 0, q - импульс продуктов распада R, θ_R - хелисити-угол R, $F_L^{(\text{cov})}$ - ковариантный фактор, $P_L^{(\text{Leg LAURA}^{++})}$ - полиномы Лежандра с изменённой нормировкой, B - амплитуда Брейта-Вигнера, $F_J^{(BW)}$ - формфактор Блатта-Вайсскопфа. Резонансы: $\phi(1020)$, $f_2'(1525)$.





Плотность вероятности

В эксперименте плотность вероятности наблюдаемых событий, соответствующая функции плотности сигнала S, включает эффективность восстановления и учитывает наличие фоновых событий. В случае, если разрешением можно пренебречь,

$$\rho(\Phi, p_S, p_B) = (1-b)\frac{\epsilon(\Phi)S(\Phi, p_S)}{\int \epsilon(\Phi)S(\Phi, p_S)d\Phi} + b\frac{\epsilon(\Phi)B(\Phi, p_B)}{\int \epsilon(\Phi)B(\Phi, p_B)d\Phi}, \quad (4)$$

где Φ - фазовое пространство, p_S - параметры плотности сигнала, ϵ - эффективность восстановления сигнала, B - функция плотности фона, p_B - её параметры и b - доля фоновых событий в сигнальной области. Функция плотности фона определена так, что она должна быть умножена на эффективность восстановления сигнальных событий, чтобы получить реальное распределение:

$$B_{\text{obs}}(\Phi, p_B) = B(\Phi, p_B)\epsilon(\Phi), \quad (5)$$

где B_{obs} - наблюдаемое распределение фона. Благодаря такому определению параметризация эффективности не нужна для выполнения подгонки.

Вычисление нормировочного интеграла

Как правило, вычисление нормировочного интеграла $\int \epsilon(\Phi) S(\Phi) d\Phi$ занимает большую часть процессорного времени. По умолчанию используется интегрирование методом Монте-Карло по событиям, сгенерированным с плотностью вероятности $d_{\mathrm{MC}}(\Phi)$ и пропущенным через моделирование детектора:

$$\int \epsilon(\Phi) S(\Phi, p_S) d\Phi = \frac{V}{N_{\mathrm{MC}}} \sum_j w_j S(\Phi_j, p_S), \tag{6}$$

где N_{MC} - число событий Монте-Карло, V - объём интегрирования, индекс j - номер события Монте-Карло и w_j - веса событий, определяемые пользователем, например

$$w_j = \frac{R(\Phi_j)}{d_{\mathrm{MC}}(\Phi_j)} = \frac{1}{d_{\mathrm{MC}}(\Phi_j)} \prod_k \frac{\epsilon^{(k)}_{\mathrm{data}}(\Phi)}{\epsilon^{(k)}_{\mathrm{MC}}(\Phi)}, \tag{7}$$

где $R(\Phi_j)$ - отношение между эффективностями в данных и Монте-Карло, определяемое из калибровочных каналов для каждой из частиц конечного состояния k.

Минимизируемая функция

Функция, которая минимизируется при небинированной подгонке методом максимума правдоподобия, равна

$$\mathcal{F} = \sum_i -2 \ln \left[(1-b) \frac{S(\Phi_i, p_S)}{\sum_j w_j S(\Phi_j, p_S)} + b \frac{B(\Phi_i, p_B)}{\sum_j w_j B(\Phi_j, p_B)} \right] - 2 \ln \frac{\epsilon(\Phi_i) N_{\mathrm{MC}}}{V}, \qquad (8)$$

где индекс i - номер события в данных. Второй логарифм в уравнении (8) постоянен и может быть опущен. Реализация в VecAmpFit также предоставляет возможность добавлять ограничения параметров. Подгонка может выполняться одновременно для нескольких наборов данных:

$$\begin{aligned} \mathcal{F} = & \sum_a \sum_i -2 \ln \left[(1-b_a) \frac{S_a(\Phi_i, p_S)}{\sum_j w_j S_a(\Phi_j, p_S)} + b_a \frac{B_a(\Phi_i, p_B)}{\sum_j w_j B_a(\Phi_j, p_B)} \right] \\ & + \sum_k C_k(p_S^{(n_k)}) + C(p_S), \end{aligned} \qquad (9)$$

где C_k - ограничения для отдельных параметров асимметричной функцией Гаусса, C - ограничивающая функция, определяемая пользователем, где a - номер точки одновременной подгонки, а функции плотности сигнала S_a и фона B_a и доля фона b_a зависят от точки одновременной подгонки.

Функция плотности сигнала

Ниже приведена функция плотности сигнала для примера $D^0 \to K^-\pi^+\pi^0$. Обратите внимание на векторное возвращаемое значение SD и директиву ACC ROUTINE для выгрузки на графические процессоры.

```
  SUBROUTINE SIGNAL_DENSITY_#MODEL#(SD, EV, PAR, PAR_BUF) BIND(C)
!$OMP DECLARE TARGET
    ! Used modules are omitted.
    IMPLICIT NONE
!$ACC ROUTINE
    REAL(C_REAL_KIND), INTENT(OUT) :: SD(VECTOR_LENGTH)
    TYPE(SIGNAL_EVENT_BUFFER_#MODEL#), INTENT(IN) :: EV
    REAL(C_REAL_KIND), INTENT(IN) :: PAR(DZ_KMPIPPIZ_SIGNAL_MODEL_PARAMETERS)
    TYPE(SIGNAL_PARAMETER_BUFFER_#MODEL#), INTENT(IN) :: PAR_BUF
    TYPE(SIGNAL_AMPLITUDES) SIG_AMP
    REAL(REAL_KIND), DIMENSION(VECTOR_LENGTH) :: M0, FF
    REAL(REAL_KIND) M1
    TYPE(COMPLEX_VECTOR) AMP_RHOP_L1, AMP_KSTM_L0, AMP_KSTM_L1, AMP_KSTZ_L0, &
&     AMP_KSTZ_L1, AMP
    INTEGER I
    ! D decay formfactors.
    CALL BLATT_WEISSKOPF_SQUARED_L1(SIG_AMP%F_D0_RHOP_KM_L1, EV%Q_D0_RHOP_KM, &
&     PAR(D0_R))
    CALL BLATT_WEISSKOPF_SQUARED_L1(SIG_AMP%F_D0_KSTM_PIP_L1, &
&     EV%Q_D0_KSTM_PIP, PAR(D0_R))
    CALL BLATT_WEISSKOPF_SQUARED_L1(SIG_AMP%F_D0_KSTZ_PIZ_L1, &
&     EV%Q_D0_KSTZ_PIZ, PAR(D0_R))
    ! The remaining code is omitted.
  END SUBROUTINE
```

Математические подпрограммы

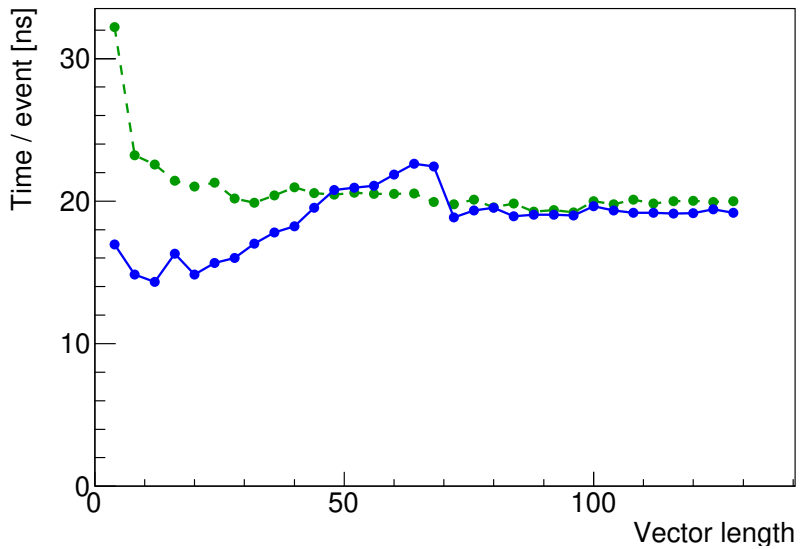
1. Константы $\sqrt{2}$, $\ln 2$, e, π, $\sqrt{\pi/2}$, $\sqrt{\pi}$, $\sqrt{2\pi}$, $\pi/180$, $180/\pi$, вычисляемые самой библиотекой при сборке с использованием вычислений с произвольной точностью.
2. Математические функции:
   - Функции Гаусса с различными не-гауссовыми хвостами, их нормировки и первообразные.
   - Полиномы Чебышёва.
   - Лямбда-функция Каллена.
   - Логистическая функция.
   - Векторизованные версии встроенных функций Фортрана `ERF`, `EXP`, `FLOOR`, `POW`.
3. Одномерная линейная интерполяция.
4. Линейная алгебра: вычисление собственных векторов и собственных значений (используя `LAPACK`).
5. Численное интегрирование: одномерное интегрирование методом Симпсона, многомерный адаптивный алгоритм Генца и Малика.
6. Тестовая статистика смешанных образцов данных, доверительные интервалы Фельдмана-Казинса.

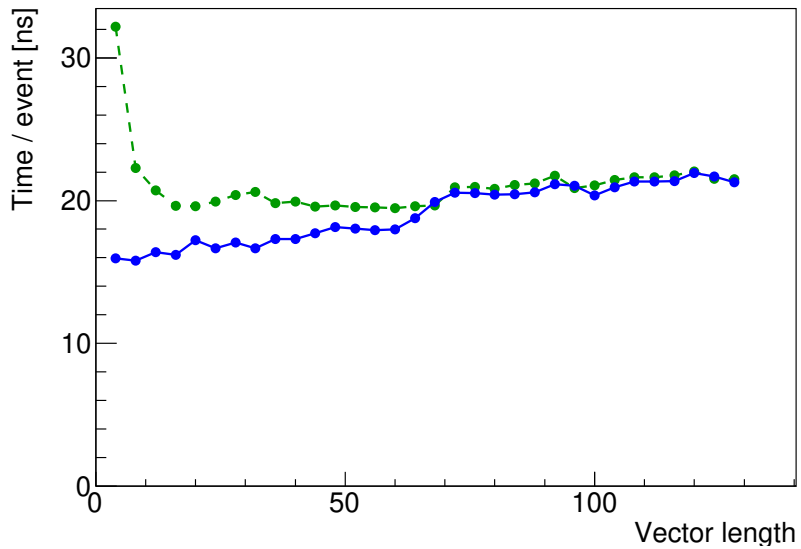
Физические подпрограммы

1. Кинематика:
   - Функции для графиков Далица: получение хелисити-угла по инвариантным массам, возможных значений квадрата инвариантной массы m_{23}^2 при заданном значении m_{12}^2, проверка, находятся ли квадраты инвариантных масс m_{12}^2 и m_{23}^2 внутри разрешённой области.
   - Импульс дочерних частиц в системе покоя материнской частицы.
   - Производные типы для трёхмерных и Лоренц-векторов.
2. Физика высоких энергий:
   - Формфакторы Блатта-Вайсскопфа.
   - Различные формы амплитуды Брейта-Вигнера.
   - Ковариантные факторы, появляющиеся в трёхчастичных ковариантных амплитудах.
   - Аналитическое продолжение импульса.
3. Квантовая механика:
   - Минимальный угловой момент для заданного распада.
   - d-функции Вигнера.

Длина вектора, оптимизация при компоновке

Производительность измеряется выполнением примера $D^0 \to K^-\pi^+\pi^0$ с одинарной и двойной точностью вычислений на двух тестовых машинах (AMD Ryzen 3600, 3.6 ГГц, 32 Гб памяти, NVIDIA GeForce GT 710 с 2 Гб памяти; Intel Core i3-10105, 3.7 ГГц, 8 Гб памяти) с Debian 12 для длин вектора, пропорциональных длине регистров AVX.





Время на событие на вызов для тестовой машины 1 (слева) и 2 (справа) для вычислений с двойной точностью. Синяя сплошная линия - время с оптимизацией при компоновке, зелёная штриховая линия - время без оптимизации при компоновке.

- Предпочитаемая длина вектора пропорциональна длине самого большого векторного регистра с небольшим целочисленным коэффициентом, зависящим от машины.
- Оптимизация при компоновке существенно улучшает производительность для оптимальной длины вектора. Её эффект уменьшается для больших длин.

Длина вектора, оптимизация при компоновке (2)

Предпочитаемая длина вектора, улучшение по сравнению со скалярной версией и эффект оптимизации при компоновке (LTO) для примера $D^0 \to K^-\pi^+\pi^0$. Время на событие на вызов приведено в наносекундах.

Машина	Тестовая машина 1		Тестовая машина 2	
Точность	Одинарная	Двойная	Одинарная	Двойная
Предпочитаемая длина вектора	40	12	24	8
Время скалярной версии	44.5	54.6	35.7	41.8
Время векторной версии	7.29	14.3	7.35	15.8
Время векторной версии без LTO	9.50	22.6	9.77	22.3
Отношение скалярного и векторного времени	6.1	3.8	4.9	2.6
Отношение времени без LTO и с LTO	1.4	1.6	1.3	1.4

Сравнение с Laura^{++}

Сравнение проводится с использованием специально предназначенного для этого примера $B^+ \to K^+K^+K^-$ с моделью, соответствующей модели GenFit3K из Laura^{++}. Создаётся такая же решётка, как в Laura^{++}, где она создаётся автоматически. Физическая модель та же, но полное соответствие вычислений невозможно: подгонка в Laura^{++} выполняет больше операций, так как в Laura^{++} проводится нормировка каждого резонанса отдельно с интегрированием по графику Далица, что не предусмотрено в VecAmpFit. Аналогично процедуре тестирования Laura^{++} генерируется 50 псевдоэкспериментов. В отличие от предыдущих тестов, полное время выполнения используется для сравнения с другими библиотеками. Используются случайные начальные значения амплитуд как для Laura^{++}, так и для VecAmpFit.

Машина	Тестовая машина 1	Тестовая машина 2
Генерация (Laura^{++})	176	135
Laura^{++}	1416	1229
VecAmpFit, длина вектора 4	321	293
VecAmpFit, длина вектора 8	317	325
VecAmpFit, длина вектора 16	318	343
VecAmpFit, длина вектора 32	343	340
VecAmpFit, длина вектора 64	341	363
Отношение (Laura^{++} / VecAmpFit)	4.5	4.2

Подгонка с использованием VecAmpFit более чем в 4 раза быстрее Laura^{++} на обеих тестовых машинах, но при этом часть этой разницы связана с другим методом нормировки в Laura^{++}, требующим больше вычислений.

Сравнение с TFA2

TensorFlowAnalysis2 (TFA2) - библиотека для амплитудных анализов, основанная на TensorFlow. Сравнение VecAmpFit и TFA2 проводится с использованием примера $D^0 \to K^-\pi^+\pi^0$ подгонкой 50 различных псевдоэкспериментов по 100000 событий с общим образцом нормировочного Монте-Карло из 1000000 событий (для GPU образцы данных и число псевдоэкспериментов уменьшены).

Тест	Время, с	Время CPU	Память
VecAmpFit	483	5151	448
VecAmpFit, свободны m	562	6062	448
VecAmpFit, свободны m, Γ	814	9060	448
TFA2, JIT	7316	20814	7589
TFA2, JIT, свободны m	9641	26825	8403
TFA2, JIT, свободны m, Γ	9788	27244	8293
TFA2, JIT, no gradient	4715	14935	1419
TFA2, no JIT	15486	174209	5108
VecAmpFit, GPU	1755	1754	310
VecAmpFit, GPU, свободны m	3216	3215	310
VecAmpFit, GPU, свободны m, Γ	6933	6933	309
TFA2, GPU	1388	1413	1619
TFA2, GPU, свободны m	1729	1757	1729
TFA2, GPU, свободны m, Γ	2198	2228	1731
TFA2, GPU, JIT	2694	2848	2172

Сравнение с TFA2 (2)

Отношения времён подгонки для TFA2 и VecAmpFit. При подгонке со свободными амплитудами, свободными амплитудами и массами, свободными амплитудами, массами и ширинами число свободных параметров равно 15, 20 и 25, соответственно.

Свободные параметры	CPU	GPU
Амплитуды	15.1	0.79
Амплитуды, m	17.2	0.54
Амплитуды, m, Г	12.0	0.32

- VecAmpFit более чем на порядок быстрее при вычислениях на центральном процессоре. Для TFA2 не наблюдается улучшения, связанного с вычислением градиента.
- На графическом процессоре, наоборот, TFA2 значительно быстрее VecAmpFit. Виден явный эффект от вычисления градиента: отношение времён подгонки для TFA2 и VecAmpFit падает при увеличении числа свободных параметров.

Заключение

1. Разработана новая библиотека VecAmpFit для выполнения амплитудных анализов.
   - Размещение данных и вычисление амплитуд векторизованы для увеличения производительности.
   - Поддерживаются параллельные вычисления при помощи OpenMP и выгрузка на графические процессоры при помощи OpenACC.
   - Возможна одновременная подгонка нескольких образцов данных со связанными амплитудами.
2. Проведены тесты для сравнения производительности VecAmpFit с существующими библиотеками, написанными в императивном подходе (Laura^{++}) и в декларативном подходе (TensorFlowAnalysis2).
   - VecAmpFit значительно быстрее обеих библиотек на CPU.
   - Однако, в случае выгрузки на GPU VecAmpFit уступает TFA2, что связано с отсутствием автоматического вычисления градиента.

Дополнительные слайды

Интегрирование на решётке

Функция правдоподобия для интегрирования на решётке аналогична интегрированию методом Монте-Карло, но вместо событий Монте-Карло используются точки решётки, а веса определены как

$$w_j = w_j^{(\text{integration})} w_j^{(\text{Jacobian})}, \tag{10}$$

где $w_j^{(\text{integration})}$ - веса метода интегрирования и $w_j^{(\text{Jacobian})}$ - якобиан преобразования между координатами фазового пространства и решётки. В примере $B^+ \to K^+K^+K^-$ веса определены как

$$w_{j_1 j_2}^{(\text{integration})} = \frac{m_{K_1^+K^-}^{(\max)} - m_{K_1^+K^-}^{(\min)}}{2} \frac{m_{K_2^+K^-}^{(\max)} - m_{K_2^+K^-}^{(\min)}}{2} w_{j_1}^{(\text{GL})} w_{j_2}^{(\text{GL})}, \tag{11}$$

где индекс $n = 1, 2$ - номер K^+, $m_{K_n^+K^-}^{(\max)}$ и $m_{K_n^+K^-}^{(\min)}$ - максимальные и минимальные инвариантные массы указанных комбинаций и $w_{j_n}^{(\text{GL})}$ - веса Гаусса-Лежандра для интегрирования в области $[1, 1]$. Решётка определена для инвариантных масс, поэтому якобиан равен

$$w_j^{(\text{Jacobian})} = \frac{\partial m^2_{K_1^+K^-}}{\partial m_{K_1^+K^-}} \frac{\partial m^2_{K_2^+K^-}}{\partial m_{K_2^+K^-}} = 4 m_{K_1^+K^-} m_{K_2^+K^-}. \tag{12}$$

Расширенная небинированная подгонка

В VecAmpFit также реализована расширенная небинированная подгонка методом максимума правдоподобия для случая без вклада фона. Плотность вероятности включает распределение числа событий:

$$\rho(\Phi, p_S) = \frac{\epsilon(\Phi) S(\Phi, p_S)}{I_S} \frac{(I_S)^{N_{\text{data}}} \exp(-I_S)}{\Gamma(N_{\text{data}} + 1)} \tag{13}$$

где $I_S = \frac{V}{N_{\text{MC}}} \sum_j w_j S(\Phi_j, p_S)$ и N_{data} - число событий в данных.

Соответствующая функция правдоподобия после отбрасывания постоянных вкладов, принятия объёма интегрирования равным 1 и добавления ограничений получается равной

$$\mathcal{F} = \sum_i -2 \ln S(\Phi_i, p_S) + 2 \frac{1}{N_{\text{MC}}} \sum_j w_j S(\Phi_j, p_S) + \sum_k C_k(p_{n_k}) + C(p). \tag{14}$$

Подгонка с явным вычислением градиента

Функция правдоподобия для интегрирования методом Монте-Карло даётся уравнением (9). Её частные производные равны

$$\frac{\partial \mathcal{F}}{\partial p_S^{(\alpha)}} = \sum_i \frac{-2(1-b)}{(1-b)\frac{S(\Phi_i, p_S)}{\sum_j w_j S(\Phi_j, p_S)} + b\frac{B(\Phi_i, p_B)}{\sum_j w_j B(\Phi_j, p_B)}} \times \left[\frac{\frac{\partial S(\Phi_i, p_S)}{\partial p_S^{(\alpha)}}}{\sum_j w_j S(\Phi_j, p_S)} - \frac{S(\Phi_i, p_S)}{\left(\sum_j w_j S(\Phi_j, p_S)\right)^2} \left(\sum_j w_j \frac{\partial S(\Phi_j, p_S)}{\partial p_S^{(\alpha)}} \right) \right] + \delta_{n_k \alpha} \frac{\partial C_k(p_S^{(n_k)})}{\partial p_S^{(\alpha)}} + \frac{\partial C(p_S)}{\partial p_S^{(\alpha)}}, \quad (15)$$

где α - номер параметра. Поскольку производные зависят от производных функции плотности сигнала, нужно вычисление этой функции и её градиента $\partial S(\Phi, p_S)/\partial p_S^{(\alpha)}$ для всех событий данных и Монте-Карло. Пользователю необходимо задать две функции: первая одновременно вычисляет функцию плотности сигнала и её градиент, вторая - только функцию плотности сигнала.

Буферы событий

Аргументы $\vec{\Phi}_{i_1}$ называются буферами событий в VecAmpFit. В примере $D^0 \to K^-\pi^+\pi^0$ буферы сигнальных событий содержат инвариантные массы двухчастичных комбинаций, импульсы распадов D^0 на промежуточный резонанс и третью частицу и импульсы распадов промежуточных резонансов на пару частиц конечного состояния:

```
TYPE, BIND(C) :: SIGNAL_EVENT_BUFFER_#MODEL#
  REAL(C_REAL_KIND) M_KM_PIP(VECTOR_LENGTH)
  REAL(C_REAL_KIND) M_KM_PIZ(VECTOR_LENGTH)
  REAL(C_REAL_KIND) M_PIP_PIZ(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_D0_RHOP_KM(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_D0_KSTM_PIP(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_D0_KSTZ_PIZ(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_RHOP_PIP_PIZ(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_KSTM_KM_PIZ(VECTOR_LENGTH)
  REAL(C_REAL_KIND) Q_KSTZ_KM_PIP(VECTOR_LENGTH)
END TYPE
```

Здесь `C_REAL_KIND` - C-совместимый KIND типа REAL (`C_FLOAT` или `C_DOUBLE`) и `VECTOR_LENGTH` - длина вектора, которые задаются на стадии конфигурации. Интерфейс C генерируется автоматически программой `vecampfit-fortran`. Суффикс `#MODEL#` - шаблонная часть, которая в примере $D^0 \to K^-\pi^+\pi^0$ всегда имеет значение `DZ_KMPIPPIZ_DEFAULT`.

Буфер параметров

Кроме буфера событий, функции плотности принимают в качестве аргумента буфер параметров. Буфер параметров - заданный пользователем производный тип, используемый для хранения значений, зависящих только от параметров подгонки. В примере $D^0 \to K^-\pi^+\pi^0$ сигнальный буфер параметров содержит массы в степени -2, комплексные амплитуды и импульсы и формфакторы распадов, соответствующие массе резонанса:

```
TYPE, BIND(C) :: SIGNAL_PARAMETER_BUFFER_#MODEL#
  REAL(C_REAL_KIND) INVERSE_M2_RHO770P
  COMPLEX(C_REAL_KIND) A_RHO770P_PIP_PIZ
  REAL(C_REAL_KIND) INVERSE_Q0_RHO770P_PIP_PIZ
  REAL(C_REAL_KIND) INVERSE_F0_RHO770P_PIP_PIZ
  REAL(C_REAL_KIND) INVERSE_F0_D0_RHO770P_KM
  REAL(C_REAL_KIND) INVERSE_Q0_KST01430Z_KM_PIP
  ! Parameters related to other resonances are omitted.
END TYPE
```

Комплексный вектор

Стандартный тип COMPLEX не подходит для векторизации, потому что действительная и мнимая части элементов комплексного массива располагаются в памяти рядом для каждого элемента. Для того, чтобы получить вектор действительных или мнимых частей, нужно выполнить перестановки данных, размещённых в памяти. Поэтому вводится тип для комплексных векторов с отдельными массивами для действительной и мнимой частей:

```
TYPE TYPE_WITH_KIND_LENGTH(COMPLEX_VECTOR)
  REAL(__KIND) RE(__LENGTH)
  REAL(__KIND) IM(__LENGTH)
END TYPE
```

где TYPE_WITH_KIND_LENGTH - макрос препроцессора C, который добавляет идентификатор типа данных и длину вектора к имени производного типа. В VecAmpFit реализованы необходимые подпрограммы для сложения, вычитания, умножения комплексного вектора и другого комплексного вектора, действительного массива, действительного числа или комплексного числа. Другие подпрограммы включают получение и установку элементов вектора, присваивание, сумму элементов, абсолютное значение, сопряжение, $-x$, $1/x$.

Прочие функции для анализа данных

Другие функции, полезные для анализа данных, включают:

1. Генерация Монте-Карло: генерация двухчастичных и многочастичных распадов с равномерным распределением по фазовому пространству.
2. Интерфейс ROOT: доступ к классам `TMinuit`, `TFile`, `TTree`, `TRandom1` из Фортрана или C.
3. Данные о частицах: массы слабо распадающихся и узких состояний из данных PDG 2020, 2022 и 2024 года.

Обработка исходного кода на Фортране

Программа `vecampfit-fortran` используется при сборке VecAmpFit. Она реализует синтаксический анализатор и несколько генераторов вывода для подмножества стандарта Фортран 2018, используемого библиотекой. Программа используется для:

1. Автоматической генерации интерфейсов к подпрограммам VecAmpFit, написанным на Фортране.
2. Автоматической генерации заголовочных файлов C для C-совместимых подпрограмм.
3. Автоматической генерации документации в формате LaTeX с использованием синтаксиса, подобного Doxygen.

Вычисления с произвольной точностью

- Библиотека для вычислений с произвольной точностью используется для точного вычисления констант при сборке VecAmpFit. Они включают математические константы и коэффициенты в рядах, используемых для вычисления векторизованных функций, таких как `VECTOR_EXP`.
- Арифметические операции: сложение, вычитание, умножение и деление реализованы с гарантированными точностью и округлением. Поддерживается округление к ближайшему числу, к нулю, от нуля, к большим значениям и к меньшим значениям.
- Есть ограничения: точность функций не гарантируется. они просто вычисляются с использованием точной арифметики подобно обычным вычислениям на CPU. Поддерживаются только точности, пропорциональные 64 битам.
- Код для арифметических операций и функций проверен сравнением результатов вычислений с библиотекой GNU MPFR. Различий не найдено. Производительность умножения и деления хуже MPFR, не рекомендуется использовать VecAmpFit как библиотеку для вычислений с произвольной точностью.