JOURNAL OF COMPUTATIONAL PHYSICS 25, 296–313 (1977)

Automation of the Nonrelativistic φ^3 Field Theory in Two Spatial Dimensions

STEVEN JAY HARRINGTON

Department of Computer Science, University of Utah, Salt Lake City, Utah 84112

AND

ALISTAIR D. C. HOLDEN

Departments of Electrical Engineering and Computer Science, University of Washington, Seattle, Washington 98195

Received December 15, 1976; revised March 10, 1977

Both symbol manipulation and numerical techniques are used to automate the calculations in a nonrelativistic φ^3 theory in two spatial dimensions. This work applies primarily to the n-loop expansion. The system performed an entire 3-loop calculation in a perturbative orderindependent fashion, that is, generated graphs and the corresponding algebraic expressions, located and manipulated divergent subgraphs, did Feynman parameterization and analytic integration of momentum variables, and did the modifications for the vertex case and the numerical integration of the Feynman parameterization integrals. Since the system is implemented in FORTRAN it can be used almost anywhere. It can be used to facilitate both research and instruction in physics.

INTRODUCTION

Both symbol manipulation and numerical techniques are used in this paper to solve problems in a nonrelativistic φ^3 field theory. Calculations of quantities in a field theory are usually done by a perturbative approach. This is because solution of the entire theory is often too difficult, while the perturbative calculations are straightforward. This method has been quite successful in quantum electrodynamics where the small coupling constant (1/137) leads to rapid convergence, and only a few terms need be considered. Unfortunately the number and complexity of the terms increases rapidly with the order of the expansion. Since the calculations are straightforward, they can be carried out on a computer using symbol manipulation techniques. Problems in quantum electrodynamics have been successfully handled by this approach for several years [3–9, 11, 14–16, 18, 20–22, 24–27, 29]. For the reader who is interested in learning more about QED calculations, there are reviews describing the theoretical calculations and the comparison of theory with experiment [2, 3, 7, 12, 19]. In addition to quantum electrodynamics, the case of the relativistic

ISSN 0021-9991

four-dimensional φ^3 theory has been considered by Calmet and Perrottet [4]. It might be noted that the symbol manipulation techniques used in these efforts need not be restricted to the solution of a specific problem, and several general algebraic manipulation programs have been developed. Among them, three which have been used for field theory calculations are ASHMEDAI, REDUCE, and SCHOONSHIP [17, 23, 28].

This work is concerned with the automation of calculations in a nonrelativistic φ^3 theory in two spatial dimensions. This theory has the advantage of having only one spinless particle. It has become of interest lately in the study of the asymptotic properties of diffraction scattering [1]. In particular the methods for calculating critical exponents used in solid state physics (e.g., the ϵ expansion, the high temperature expansion, and the n-loop expansion) have been applied to Gribov's Reggeon Calculus.

This work applies primarily to the n-loop expansion. Calculations have been carried out to second order in the coupling squared (2-loop) [13]. The slow convergence requires the calculation of higher order terms. It is the purpose of this work to demonstrate how a computer may be used to perform the entire calcualtion in a perturbative order-independent way. This controls the problem of increasing complexity, and allows a 3-loop calculation. Higher order calculations, at the present state of technology, appear to be limited by computation time. Since this work is intended to demonstrate how the calculation can be handled by machine, simplicity is chosen over efficiency.

The elements of the solution consist of the generation of graphs, generation of the corresponding algebraic expression, location and handling of divergent subgraphs, Feynman parameterization and the analytic integration of momentum variables, the modifications for handling the vertex case, and the numerical integration of the Feynman parameterization integrals.

THEORY

A detailed statement of the problem follows. The free Lagrangian is defined as

$$L_0 = \frac{i}{2}\,\phi^+ \overleftrightarrow{\frac{\delta}{\delta t}}\,\phi - a_0\,\nabla\phi^+\nabla\phi - M_0\phi^+\phi \tag{1}$$

where $\phi = \phi(x, t)$ is the unrenormalized field written as a function of a two-dimensional space vector x, conjugate to momentum K, and time t conjugate to energy E. The full Lagrangian density is

$$L = L_0 - (i/2)\,r_0(\phi^+\phi^2 + \phi^{+2}\phi). \tag{2}$$

The Green functions are

$$G^{(N,M)}(\mathbf{x}_i\,,\,t_i\,;\,\mathbf{y}_j\,,\,\tau_j) = \prod_{i=1}^{N}\prod_{j=1}^{M}\langle 0 \mid T\phi^+(\mathbf{y}_j\,,\,\tau_j)\;\phi(\mathbf{x}_i\,,\,t_i)\mid 0\rangle \tag{3}$$

and their Fourier transform is given by

$$\delta\left(\sum E\right)\delta^2\left(\sum \mathbf{K}\right) G^{(N,M)}(E_i, K_i)$$

$$= \int \prod_{i=1}^{N} d^2x_i\, dt_i \exp(i(E_i t_i - \mathbf{K}_i \cdot \mathbf{x}_i))$$

$$\times \prod_{j=1}^{M} d^2y_j\, d\tau_j \exp(-i(E_j\tau_j - \mathbf{K}_j \cdot \mathbf{y}_j))\, G^{(N,M)}(\mathbf{x}_i, t_i; \mathbf{y}_j, \tau_j). \tag{4}$$

In this work, only the 2- and 3-point unrenormalized connected proper vertex functions $\Gamma(1, 1)$ and $\Gamma(1, 2)$ are considered. They are defined by

$$\Gamma^{(N,M)}(E_i, K_i) = \prod_{l=1}^{N+M} (G^{(1,1)}(E_l, K_l))^{-1}\, G^{(N,M)}(E_i, K_i) \tag{5}$$

and are the connected Green functions with the external legs amputated. The program will make the following calculations:

$$\begin{aligned} &-\frac{\partial}{\partial E}\,\Gamma^{(1,1)}(E, \mathbf{K})|_{E=-E_N, K^2=0}\,; \\ &-\frac{\partial}{\partial K^2}\,\Gamma^{(1,1)}(E, \mathbf{K})|_{E=-E_N, K^2=0}\,; \\ &\Gamma^{(1,2)}(E_i, \mathbf{K}_i)|_{E_1=2E_2=2E_3=-E_N, \mathbf{K}_i\mathbf{K}_j=0}\,; \end{aligned} \tag{6}$$

where E_N is some normalization energy. In two dimensions, elementary closed loops are divergent; mass counter terms are therefore required. This partial renormalization is given by

$$\Gamma^{(1,1)}(E, K)|_{E=M=0, K^2=0}\,. \tag{7}$$

The calculation of $G(N, M)$ follows from a set of Feynman rules:

(1) draw all topologically and time order distinct graphs;
(2) calculate the following integral for each loop: $\int d^2K$;
(3) conserve energy and momentum at each vertex;
(4) multiply by $r_0/2\pi$ for each vertex;
(5) multiply by the expression $i/(\sum_{\text{lines cut}} E_i - a'K_i^2 - M_i)$ for each time cut;
(6) multiply by $\frac{1}{2}$ for each elementary closed loop.

The program described in the next sections follows, for the most part, the same procedures as in a human calculation of the functions [10]. All graphs are generated, an analytic expression is created from the Feynman rules, the expression is Feynman parameterized and evaluated. The momenta integrals are done analytically while the Feynman parameterization integrals are done numerically.

GRAPH GENERATION

The first step is the generation of a graph. In the relativistic theories, where only topologically distinct graphs are considered, this is often done recursively, making all possible modifications of the lower order graphs and removing any redundancy [24, 26]. However, for the nonrelativistic case the energy integration can be replaced if the time order of the vertices is taken into account. The algorithm given below uses this to calculate the graphs of a given order directly. It is not necessary to determine the lower order graphs and insert them into each other. Each graph is unique, so that it is not necessary to check for duplicates.

Each graph represents a possible interaction sequence between identical particles. In the case of the propagator $\Gamma(1, 1)$, there is only one particle when the sequence begins and ends. Since $\Gamma(1, 1)$ and not $G(1, 1)$ is considered, at no other time will there exist only a solitary particle. The interaction allows two possible bare vertices, and these are used to construct the graph. The first (type 1) causes one particle to divide into two particles, while the second (type 2) combines two particles into a single particle. The propagator must be composed of equal numbers of type-1 and type-2 vertices. For an N-loop calculation, propagator graphs are composed of $2N$

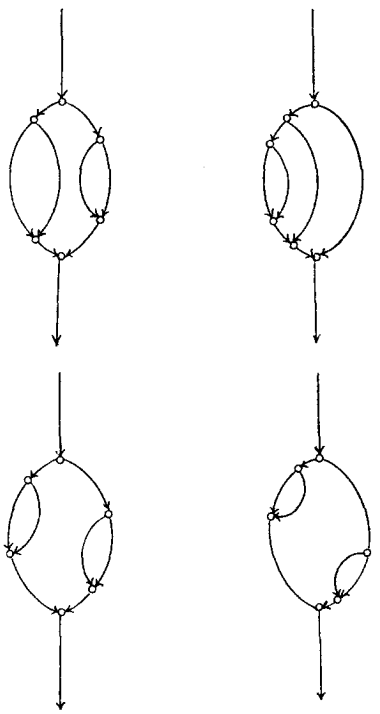


FIG. 1. Four distinct graphs. Both topological form and the order which the vertices appear are important.

vertices. Graphs are distinct if they are topologically different, or if the sequence in which the vertices are attached differs. One may think that each vertex, in the sequence composing a graph, occurs at some time with no two occurring at the same time. An equivalent conception is to make all possible cuts on the topologically distinct graphs such that each piece has one and only one vertex, and no two cuts intersect. These cuts divide the graph into time increments. For example, all graphs shown in Fig. 1 are considered different. Time proceeds from the top to the bottom of the graph.

Data Structure

The data structure used in generating the graphs is a one-dimensional array of integers. This array is partitioned into cells (Fig. 2). Each cell corresponds to the state of the interaction at some time between vertex events. It tells how many of each type of vertex remain to be used, which particle lines are present, and at which vertices they were created. This is done using the following representation. Each vertex is given a number from 1 to N, where N is the number available in a given order. Type-1 vertices are given odd numbers, type-2 even numbers. Vertices are used in order within type. The first two cell elements give the negative of the next odd, and the next even vertices to be used. The negative numbers delimit the cell. They are followed by one element for each line; the element contains the number of the vertex from which the line originated. Since not all cells generated will correspond to a single graph, there is also an array of pointers which will point to those cells which compose the graph. The method used for generating all graphs is the following. Given a cell, generate all possible cells corresponding to the next time or cut (i.e., after a vertex has been used). Pick one of the new cells and repeat the operation. Continue until there are no more vertices available. For calculations of order higher than 1-loop, the first two cells are fixed and can be provided in the initialization. The algorithm for generating all graphs is now presented.

(1) Initialize by entering the first two cells. Store a pointer to the first cell for TIME = 1. Set TIME = 2. Consider the last cell generated.

(2) Given a cell, store a pointer to that cell for the given TIME. Set TIME to TIME + 1.

(3) Test to see if there are any type-1 vertices available; if not go to 9.

(4) Consider the first line of the cell.

(5) Given a line, enter a new cell, with two lines from the type-1 vertex replacing the line considered.

(6) Consider the next line.

(7) If it comes from the same vertex as the preceding line, go to 6.

(8) If not out of lines, go to 5.

(9) Test to see if there are any type-2 vertices available. If not, go to 15.

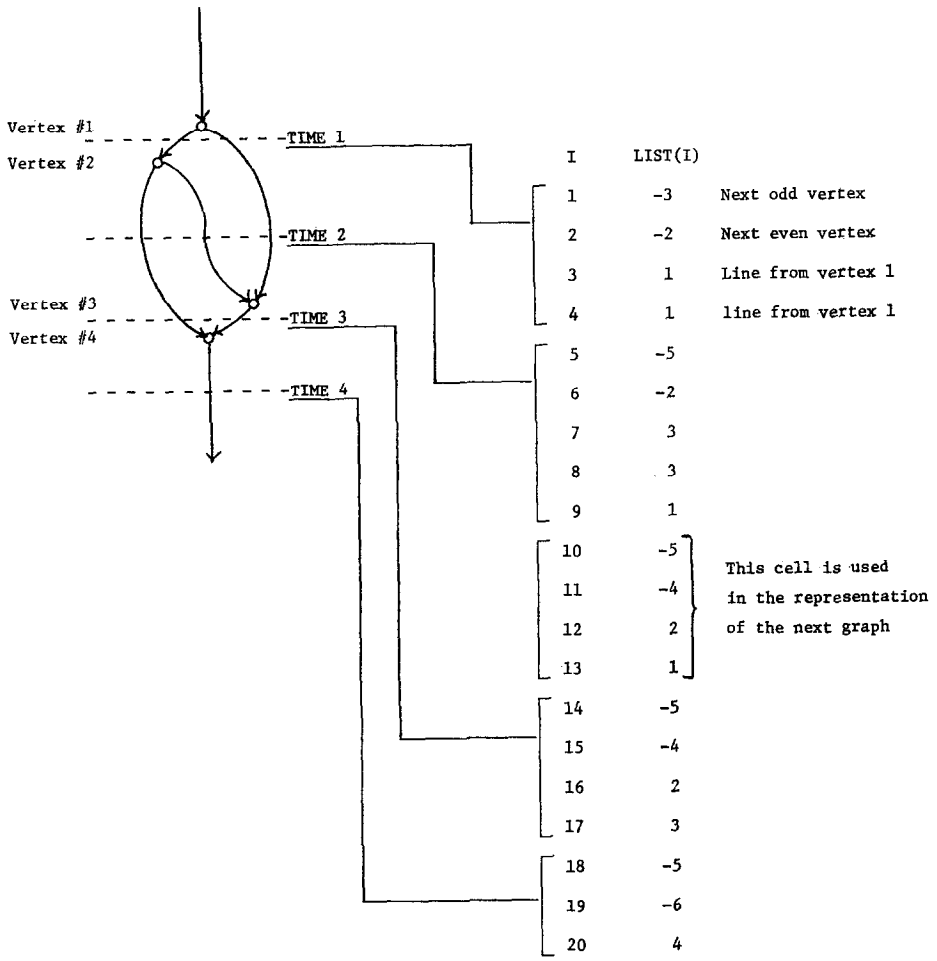


FIG. 2. Example of the internal representation of a graph and the data structure for graph generation.

(10) Consider the first line to be line 1. Line 2 is the next line. If line 2 is the last line and there is more than one type-2 vertex left, go to 16.

(11) Enter a new cell with a line from the type-2 vertex replacing line 1 and with line 2 deleted.

(12) Increment line 2, if not out of lines, go to 11.

(13) Increment line 1. If the new line 1 comes from the same vertex as the old line 1, go to 13.

(14) Set line 2 to the line following line 1. If out of lines, consider the last cell entered and go to 2, otherwise go to 11.

(15) Store a pointer to the last cell. A graph has now been generated. Its lines

for any TIME are given by the cell located by the pointer stored for that TIME. The vertex inserted between TIME $= i$ and TIME $= i + 1$ can be determined by noting changes in the cells pointed to for those times. Further processing of the graph is done at this point.

(16) Delete the last cell and decrement TIME by 1. If there are no more cells, stop since all graphs have been generated.

(17) If the new last cell is pointed to for the given TIME, go to 16. Otherwise, consider the last cell and go to 2.

An example of the structure for the first graph of the 2-loop propagator is shown in Fig. 2.

The program prints a picture of the graph being considered using the "/" and vertical bar "|". The graph is directed from the top to the bottom of the page

Energy Denominators

Each graph has a corresponding algebraic expression which consists of a set of integrals over the product of denominators. The denominators are of the form $[E + \sum_i a'\mathbf{K}_i^2 + M_i]$, where there is one denominator for each time, and K_i and M_i are the momenta and masses associated with each line which exists at the time. The expression for the entire graph is kept in the Feynman parameterized form. At this point, it is only necessary to represent one denominator at a time. The denominator can then be Feynman parameterized. The momenta of the denominators are represented by a two-dimensional array, $D(j, i)$, with the first index giving the line of the graph and a second, the momentum. The coefficients of the momenta on the line (-1, 0, or 1) are stored. Their sum is not squared at this point. All lines have the same mass so that the coefficient of the mass term is just the total number of lines for the time cut. E is the external energy entering the graph and is constant, identical for all times. The denominator for a given time cut is constructed from the denominator for the preceding time cut and the cells representing the graph for the two times. A momentum variable must be present for each closed loop of the graph. This is easily done by adding a new momentum at each type-1 vertex. Momentum is conserved at each vertex. The first two denominators, like the first two cells in the graph generation, may be written immediately. The first is

$$\begin{matrix} 1 & 1 & 0 & 0 & \cdots \\ 0 & -1 & 0 & 0 & \cdots \end{matrix}$$

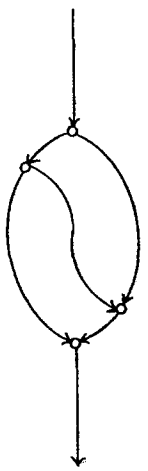
with a mass coefficient of 2. The first column represents the external momentum entering the graph; the second is the new momentum corresponding to the new vertex. The representation of each denominator is shown in Fig. 3.

Given a denominator, the rules for representing the next denominator are as follows:

(1) Compare the cells representing the two times, line element by line element, until they differ. If the differing element in the later cell is even, go to 3.

(2) A type-1 vertex was inserted here, so move all following rows down 1 (the value of $D(j, i)$ is moved to $D(j+1, i)$) to make room for the new line. Change the coefficient of the next momentum ($\mathbf{K}_l$) to be used from 0 to +1 in the denominator row corresponding to the different line element. The following row represents the new line. Because of momentum conservation, it must have 0 for all coefficients except that for $\mathbf{K}_l$. This coefficient is -1. Increase the mass coefficient by 1. The representation is finished.

(3) A type-2 vertex was used so that the row of the denominator representation corresponding to this line element (row A) will be modified and some later row (row B) will be deleted. Continue comparing the graph cells until a second dis-



$$[\ E + 2M + a(\ (\vec{K}+\vec{K}_1)^2 + \vec{K}_1^{\,2})\ ]$$

$$\begin{matrix} 1 & 1 & 0 \\ 0 & -1 & 0 \end{matrix}$$

$$[\ E + 3M + a(\ (\vec{K}+\vec{K}_1+\vec{K}_2)^2 + \vec{K}_1^{\,2} + \vec{K}_2^{\,2})\ ]$$

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Energy Denominator $\longrightarrow$ $[\ E + 2M + a(\ (\vec{K}+\vec{K}_2)^2 + \vec{K}_2^{\,2})\ ]$

Momentum Array $\longrightarrow$

$$\begin{matrix} 1 & 0 & 1 \\ 0 & 0 & -1 \end{matrix}$$

FIG. 3. Example of data structure for energy denominators. Both the external form of the energy denominators and the internal representation of momentum contributions are shown for the three time cuts of the graph.

crepancy is found. This corresponds to the row to be removed (row B). Add the coefficients of row B columnwise to row A. Move all rows following row B up one row (the value of $D(j, i)$ becomes what used to be in $D(j + 1, i)$. Decrease the mass coefficient by 1. The representation is finished.

As each denominator is completed, it is printed so that the program presents the symbolic expression for the integrand.

Feynman Parameterization

Feynman parameterization is the application of

$$d_1^{-N_1} d_2^{-N_e} \cdots d_i^{-N_i} = \frac{\Gamma(N_1 + N_2 + \cdots + N_i)}{\Gamma(N_1)\,\Gamma(N_2) \cdots \Gamma(N_i)} \int_0^1 dx_1 \int_0^1 dx_2 \cdots \int_0^1 dx_i \times \delta\left(1 - \sum x_i\right) x_1^{N_1 - 1} x_2^{N - 1} \cdots x_i^{N_i - 1}\, g(N_1 \cdots N_i), \tag{8}$$

where

$$g(N_1 \cdots N_i) = (x_1 d_1 + x_2 d_2 + \cdots + x_i d_i)^{-(N_1 + N_e + \cdots + N_i)}. \tag{9}$$

This is applied to the expression derived from the graph, where the d's are the denominators found for each time, i.e., d is of the form $[E + \sum_j a' \mathbf{K}_j^2 + M_j]$. The reason for doing this is that now the K terms are found only in a sum. Since the K's are integrated over all space, translation by the substitution $\mathbf{K} \to \mathbf{K} + \sum_l b_l \mathbf{K}_l$ has no effect on the answer. Therefore the cross terms $\mathbf{K}_i \cdot \mathbf{K}_j$ in the expression may be removed by completing the square. The K integrals may now be carried out. The angular part just gives 2π and the radial part is of the form

$$\int_0^\infty dK^2 (AK^2 + B)^{-m} = (m - 1)\, A^{-1} B^{-m}. \tag{10}$$

This method is particularly useful in nonintegral demension space. The $\{x_i\}$ integrals must still be done, of course, but they are scalar integrals with a finite range of integration.

A three-dimensional array F is used to represent the momentum part of the Feynman parameterized expression. $F(i, j, m) = c_{ijm}$ holds the coefficient of the $c_{ijm} \mathbf{K}_i \cdot \mathbf{K}_j X_m$ term. Once an energy denominator has been determined, the coefficients of the $\mathbf{K}_i \cdot \mathbf{K}_j$ products may be calculated and entered into the matrix. A different third index m is given for each denominator. A one-dimensional array $FM(m)$ holds the coefficients of the mass terms. An example of the F and FM arrays is given in Fig. 4. The algebraic expression for the Feynman parameterized form of the integrand is printed by the program.

Once the contributions from all denominator terms have been added to the F matrix, the integration can proceed. Given values for the X variables the integrand

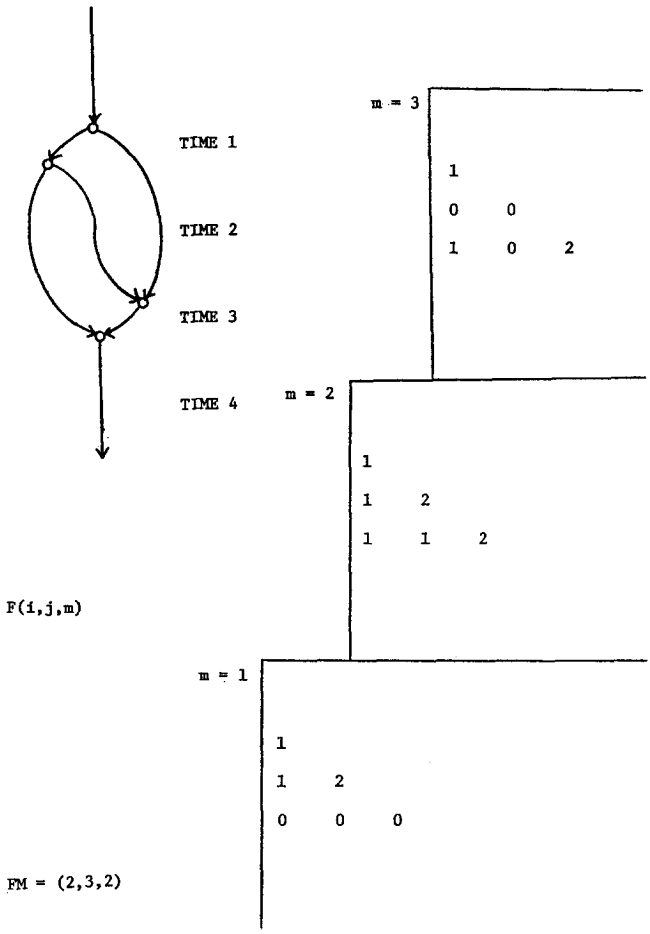


FIG. 4. Example of data structure for Feynman Parameterized form. A three-dimensional array is used to hold the coefficients of the momentum factors for each of the contributing energy denominators.

must be calculated. The first step is to create a two-dimensional array F' representing the coefficients of the K's for the given values of the X variables, i.e., $F'(i,j)$ is set to $\sum_m F(i,j,m)\, Xm$. The next step is to complete the square for the K variable. That is F' will be transformed so that it holds the coefficients of a new set of K variables such that only diagonal elements contribute. The transformation procedure which diagonalizes the matrix is as follows. For each momentum m for all $i, j < m$, $F'(i,j)$ is set to $F'(i,j) - F'(m,i)^* F'(m,j)/F'(m,m)$ where m begins at the final momentum and steps back to the second momentum. This leaves off-diagonal elements nonzero, but they are simply not used. At this point the momentum integrations can be done; which means dividing the remaining factor by the diagonal F' elements (it is not divided by $F'(1,1)$ since this is the coefficient to the external momentum). The remaining

factor is the energy, mass, and external momentum contribution. The result is the integrand for the Feynman parameter integrations. For the propagator the $-\partial/\partial E$ and $-\partial/\partial K^2$ derivatives are desired. However, since after having done the internal K integrals the expression is of the form

$$(F'(2, 2)\, F'(3, 3) \cdots)(cE + dM + F'(1, 1)\, K^2)^{-j},$$

the derivative can be done immediately by multiplying by a coefficient and changing the exponent.

Divergences

There is unfortunately a complication to the described procedure, which is that some of the graphs are divergent. Simple power counting shows that graphs diverge when they have elementary closed loops (i.e., low order propagators) which have no vertices occurring during the loop's existence. In Fig. 5 graphs a and c diverge while b is finite. In order to render these graphs finite, a subtraction of mass counter terms is necessary. The subtracted term may be graphically represented by replacing the loop with a $\times$ (Fig. 6). The energy denominator terms corresponding to the $\times$ are equal to those of the propagator it replaces with external momenta set to zero and the external energy equal to minus the mass (see Eq. (7)). However, the subtraction alone may still not give a well-defined expression; analytic continuation is necessary. This can be done on the Feynman parameterized form by the following regularization procedure. Assume there are some denominators d_i, a loop term $(t)^N$, and its mass counter term $(c)^N$. Then

$$\left(\prod_i d_i^{-m_i}\right)(t^{-N} - c^{-N}) \tag{12}$$

is to be considered. It may be written as

$$\lim_{\sigma \to -1} \sum_{j=0}^{N-1} \left(\prod_i d_i^{-m_i}\right) t^{j-N} c^{-j-1} (c - t)^{-\sigma}. \tag{13}$$

The Feynman parameterization gives

$$\lim_{\sigma \to -1} \sum_{j=0}^{N-1} \frac{\Gamma(\sum_i m_i + \sigma + N + 1)}{\prod_i \Gamma(m_i)\, \Gamma(\sigma)\, \Gamma(N - j)\, \Gamma(j + 1)} \prod_i \int_0^1 dx_i \int_0^1 dy_1 \int_0^1 dy_2 \int_0^1 du$$

$$\delta\left(1 - \sum x_i - y_1 - y_2 - u\right) x_i^{m_i - 1} y_1^{N-j-1} y_2^{j} u^{\sigma - 1} \tag{14}$$

$$\left(\sum_i x_i d_i + (y_1 - u)\, t + (y_2 + u) c\right)^{-(\sum m_i + \sigma + N + 1)}$$

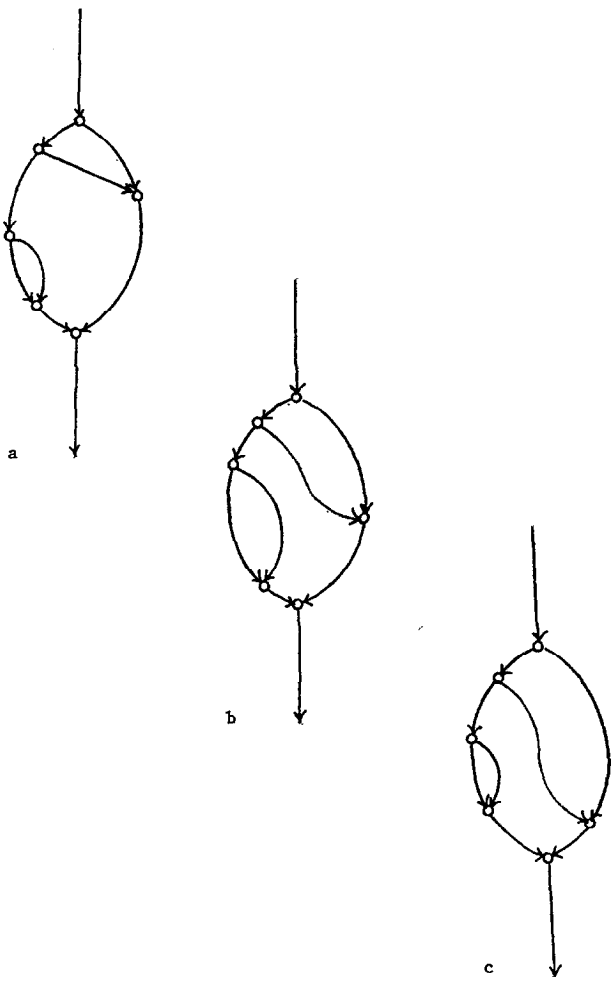


FIG. 5. Divergent and nondivergent graphs. Graphs a and c are divergent while b is not.

and the analytical continuation gives

$$-\frac{\partial}{\partial u}\Big|_{u=0} \frac{\Gamma(\sum_i m_i + N)}{\prod_i \Gamma(m_i)\,\Gamma(N)} \prod_i \int_0^{1-u} dx_1 \int_0^{1-u-x_1} dx_2 \cdots \int_0^{1-u-\Sigma x} dy_1\, x_i^{m_i-1}$$

$$\delta\Big(1-u-\sum_i x_i\Big)^{N-1} \Big(\sum_i x_i d_i + (y_1-u)\,t + \Big(1-\sum_i x_i - y_1\Big)c\Big)^{-(\sum m_i+N)} . \tag{15}$$

Therefore, a differentiation is necessary for each closed loop containing no external cuts. This is done numerically, and doubles the time necessary for the calculation. An additional Feynman integration variable also has been picked up. However, for the divergent loops the denominator at the time just before the loop is identical

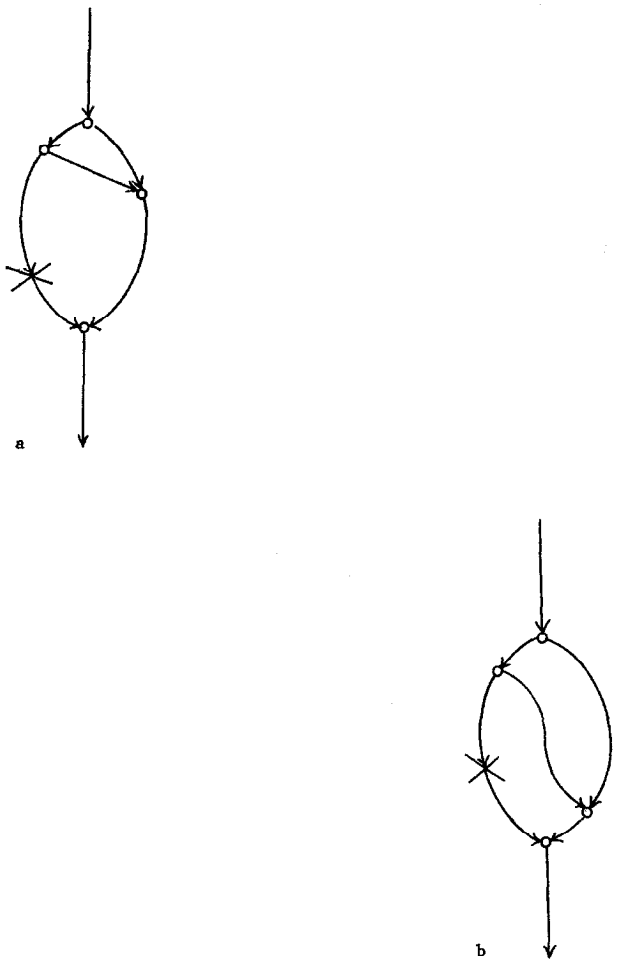


FIG. 6. Graphs representing mass counter terms. These correspond to graphs a and c of Fig. 5.

to the denominator at the time just after the loop. The denominators may be combined, reducing the number of integrations by one. Thus there is no net change in the number of integrations. There is, however, a change in the integration limits, the effects of which will be mentioned later.

VERTICES

Earlier it was stated that both 2- and 3-point functions were to be calculated. However, up to this point, only the propagator has been discussed. In this section the changes which must be made in order to handle vertices are described. The first step is to generate them. A set of graphs, which include all of the vertex graphs for the

n-loop order, is the set of all propagator graphs for the $(n+1)$-loop order, with the final joining vertex removed. This set also contains graphs with closed loops on the external legs. These graphs should not be included. They are easily distinguished. The program already checks for loops. If a loop occurs on the external vertex the graph is deleted. In generating the propagator graphs, flipping the graph about a line through the external legs produced an equivalent graph. To remove this symmetry the first two bare vertices were initially specified. However, in the vertex case, the flipped graphs are distinct and should be included. But since they are calculated at a symmetric point the flipped graphs contribute the same amount as the unflipped graphs. The program therefore calculates only half of the 3-point graphs. The full answer is just twice the number calculated.

The next problem is that one extra momentum was described which results in the external legs leaving the vertex having nonzero momentum. To correct this, one of the momenta contributing to that in the external legs must be redefined in terms of the other momenta so that their sum is zero. What this means to the program is some shuffling of the coefficients stored in the F array. Since the expression for the energy denominators is printed as they are calculated, it will contain this error. The Feynman parameterized expression, however, will be correct.

Finally, there is an additional time cut, which would imply another Feynman parameter integration. There is, however, an alternative for the case where external momenta are zero. For this case the square completion need not be done on the first internal momentum K_1, and the first time need not have a Feynman parameter attached, so that the form

$$\int_0^\infty dk_i^2(E+2M+2K_1^2)^{-i}\,(AK_1^2+B)^{-j} \tag{16}$$

is obtained. This form can be integrated without combining

$$B^{-j}(E+M)/(i+j-1)\,{}_2F_1(1,j,i+j;1-(E+M)A/B). \tag{17}$$

It was found that for this program, evaluating the hypergeometric function was faster than doing an additional Feynman parameter integration.

Numerical Integration

The numerical integration algorithm will be discussed only briefly. As has been indicated, the Feynman parameterization leaves integrals of the form

$$\int_0^1 dx_1 \int_0^1 dx_2 \cdots \int_0^1 dx_N \delta\left(1-\sum x_i\right) f(x_i). \tag{18}$$

Doing the last integral changes the limits

$$\int_0^1 dx_1 \int_0^{1-x_1} dx_2 \cdots \int_0^{1-x_1-x_2\cdots-x_{n-2}} dx_{N-1}\, f(x_i) \tag{19}$$

where

$$x_N = 1 - \sum_{i=1}^{N-1} x_i . \tag{20}$$

For simplicity, while allowing arbitrary N, a rectangular integration was used with constant step size. It may be noted that this integration scheme is neither very efficient nor accurate.

Although the integral is finite, the integrand may diverge at the limits. A similar problem arises in QED calculation, and work has been done toward developing. appropriate integration schemes. One approach is an adaptive Monte Carlo routine, where the integration is divided into subvolumes, and the variance of the subvolume integral is used to guide further subdivisions [18]. A second approach is to use Gaussian quadratures after a mapping is performed to smooth the integrand [22].

If counter terms are needed, two complications arise: First, there will be a numerical derivative d/du which entails calculating at two values of u, integrals of the form

$$\int_0^{1-u} dx_1 \cdots \int_0^{1-u-x_1-\cdots-x_{n-2}} dx_{N-1} f(x_i , u). \tag{21}$$

u may be chosen to be $U1 =$ zero, and some other point smaller than the smallest X value, i.e., $U2 < dX/N$. The different integration limits mean that two different step sizes $dX1$ and $dX2$ must be maintained. The integrals are calculated simultaneously to avoid the loss of significant figures.

The second complication is that Feynman parameterization must be applied to each internal closed loop to form it into a single denominator term before the subtraction and regularization can be carried out. Thus there may be several series of integrals,

$$\int_0^{1-u} dx_1 \cdots \int_0^{1-u_1-x_1-\cdots-x_{M-2}} dx_{M-2} \int_0^{1-u_2} dx_{M+1} \cdots \int_0^{1-u_2-x_{M+1}\cdots-x_{N-2}} dx_{N-1} \cdots, \tag{22}$$

although the total number of integrations does not increase. The limits have changed, so that under the constant step size algorithm, the calculation of expression (22) may take appreciably longer than that of (19).

The integration procedure increments the innermost integration parameter from zero to its upper limit. When the upper limit is reached, the integration parameter is reset to its lowest value, and the parameter for the next integral is incremented. The incrementing continues in this manner until the outermost integral is stepped past its upper limit at which point the integration is completed. This procedure applies even when there are groups of integrals (due to counter terms) as in expression (22). The program does need to know the value of the upper limit, but this is just a subtraction of variables in the integral group. The only new information which must be supplied when handling groups is with which variables the group begins and ends. This is supplied by means of a list of those variables which begin integration groups.

COMPUTATION TIME AND MEMORY

For the purpose of discussing the time and memory requirements for this program, the processing will be divided into two areas: First, the graph generation, and second, the graph processing. The requirements for graph generation were determined empirically from generation of the 1-loop to 5-loop propagator. It was found that the space required to generate the graphs approximately doubled with each order. Processing of the graphs requires several arrays, the largest of which was three dimensional and grew as the order cubed. Therefore, this N-loop expansion method requires approximately $2^N + N^3$ memory. It should be noted that it is not necessary to save representations of all possible configurations for the next time cut, as was done by the program. By saving only the type of vertex and the line to which it was attached, the representation of the graph cell need not be generated until it is actually used. Therefore the space requirements for graph generation could be reduced to order $N \log(N)$.

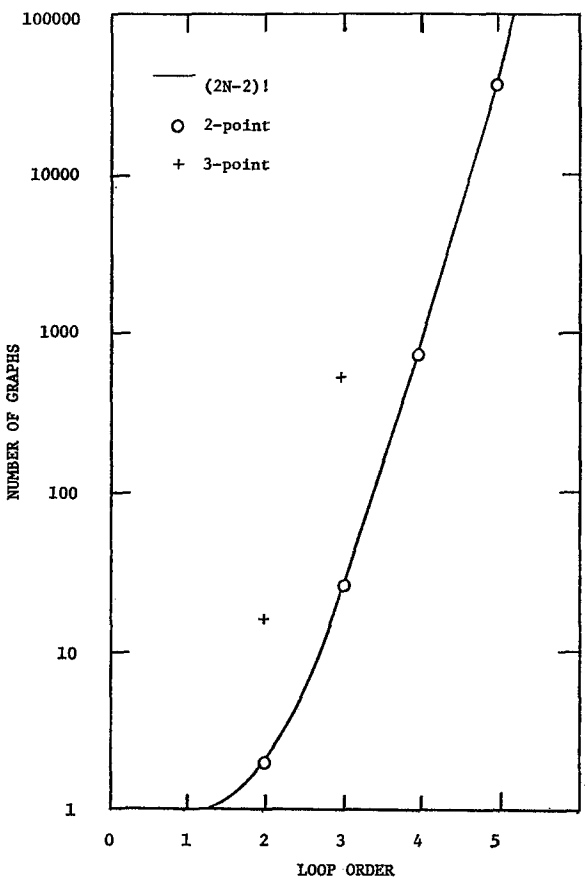


FIG. 7. The number of graphs vs loop order.

Although the space requirement grows exponentially, it is the time requirement which limits the expansion. The number of graphs grows approximately as $(2N-2)!$ for the propagator, and as is proportional to $(2N)!$ for the vertex (Fig. 7). Processing each graph requires $(2N-2)$ nested integrations. The time necessary for the 3-point functions grows as $(2N)!\, b^{2N}$. In the light of this it would seem that an extraordinary effort would be necessary to proceed beyond the 3-loop calculation.

Conclusions

The procedure which has been described is capable of generating all graphs of any given order, and of parameterizing and evaluating any graph in the theory. The two phases of processing are independent and could be separated so that, for example, individual graphs of any order could be calculated. The scheme is sufficiently powerful to extend the evaluation of the 2- and 3-point function from second order to third order, but further extension of order appears unlikely because of the inordinate computation time required. The greatest need for increased efficiency lies in the numerical integration. This may be facilitated by considering, for example, a specific perturbative order, rather than arbitrary order.

The program could be generalized to evaluate the 2- and 3-point functions at arbitrary values of external energy and momentum. The program described here allows only variations in the energy (Eq. (6)). A less straightforward improvement would be to take full advantage of the symmetries occurring when the external energy is zero.

Another possible extension of the program would be a generalization from two to D spatial dimensions. The subtraction of counterterms inside the integral allows just such a generalization. With $D = 4$, the vertex subgraphs become divergent and further subtractions are necessary. But for $D < 4$ the expressions are finite, although converging less rapidly for increasing D. On the other hand, if all calculations are to be done in two spatial dimensions, the nondivergent counter terms could be handled more efficiently. This suggests using extrapolation techniques to calculate a third order ϵ expansion. Finally, functions other than the 2- and 3-point functions could be calculated by beginning with a representation for the correct number of incoming particles, ending with the correct number, and then removing any graphs with external propagators. This is just what was done to generate the 3-point function. The calculation of the 3-point function could be improved by calculating the type-2 rather than the type-1 vertex; that is, having two external legs enter the vertex and one leg leave. This would ensure that all external momenta would be zero, and thus remove the need for renaming the momenta as is done by this program.

References

1. H. D. I. Abarbanel, J. B. Bronzan, R. L. Sugar, A. R. White, *Phys. Rep. C* **21** (1975), 119.
2. S. J. Brodsky and S. D. Drell, *Ann. Rev. Nucl. Sci.* **20** (1970), 147.

3. S. J. BRODSKY, "Second Colloquium on Advanced Computing Methods in Theoretical Physics" (A. Visoonti, Ed.), Marseilles, France, 1971.
4. J. CALMET AND M. PERROTTET, *J. Comput. Phys.* 7 (1971), 191.
5. J. CALMET, *Comput. Phys. Commun.* 4 (1972), 199.
6. J. CALMET AND A. PETERMAN, *Phys. Lett. B* **47** (1973), 369.
7. J. CALMET, "Third International Colloquium on Advanced Computing Methods in Theoretical Physics" (A. Visoonti, Ed.), Marseilles, France, 1973.
8. J. CALMET, *SIGSAM Bull. ACM* **8** (1974), 74.
9. J. A. CAMPBELL AND A. C. HEARN, *J. Comput. Phys.* **5** (1970), 280.
10. J. S. R. CHISHOLM, *Proc. Cambridge Philos. Soc.* **48** (1952), 300.
11. J. S. R. CHISHOLM, *J. Comput. Phys.* **8** (1971), 1.
12. F. COMBLEY AND E. PICASSO, *Phys. Rep. C* **14** (1974), 1.
13. J. W. DASH AND S. J. HARRINGTON, *Phys. Lett. B* **51** (1975), 249.
14. J. A. FOX AND A. C. HEARN, *J. Comput. Phys.* **14** (1974), 301.
15. A. C. HEARN, *Commun. ACM* **9** (1966), 573.
16. A. C. HEARN, *Commun. ACM* **14** (1971), 511.
17. A. C. HEARN, "Third International Colloquium on Advanced Computing Methods in Theoretical Physics" (A. Visoonti, Ed.), Marseilles, France, 1973.
18. B. E. LAUTRUP, "Second Colloquium on Advanced Computing Methods in Theoretical Physics" (A. Visoonti, Ed.), Marseilles, France, 1971.
19. B. E. LAUTRUP, A. PETERMAN, AND E. DE RAFAEL, *Phys. Rep. C* **3** (1972), 197.
20. M. J. LEVINE, *J. Comput. Phys.* **1** (1967), 454.
21. M. J. LEVINE AND J. WRIGHT, *Phys. Rev. Lett.* **26** (1971), 1351.
22. M. J. LEVINE AND J. WRIGHT, Second Colloquium on Advanced Computing Methods in Theoretical Physics (A. Visoonti, Ed.), Marseilles, France, 1971.
23. M. J. LEVINE AND R. ROSKIES, "Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation" (R. D. Jenks, Ed.), Yorktown Heights, New York, 1976.
24. J. PALDUS AND H. C. WOND, *Comput. Phys. Commun.* **6** (1973), 1.
25. A. PETERMAN, *Phys. Lett. B* **35** (1971), 325.
26. D. C. RAPAPORT, *Comput. Phys. Commun.* **8** (1974), 320.
27. C. SCHWARTZ, *J. Comput. Phys.* **3** (1969), 512.
28. H. STRUBBE, Proceedings of EUROSAM '74, *SIGSAM Bull.* **8** (1974), 55.
29. S. M. SWANSON, *J. Comput. Phys.* **4** (1969), 171.