Monte Carlo Simulation in Euclidean Time for Scalar Boson Quantum Field Theory with Quartic Self-Coupling

Baran Bodur
Department of Physics, Duke University, Durham, NC, USA, 27708

I. Lattice Action and Mathematical Derivations

We will study the scalar boson QFT with the Euclidean lattice action given as in equation 1.

$$S_E^l = a^d \sum_x \left\{ \frac{1}{2}(\frac{2d}{a^2} + m_0^2)\phi_x^2 - \frac{1}{a^2}\sum_\mu \phi_x\phi_{x+\mu} + \frac{\lambda_0}{4!}\phi_x^4 \right\} \quad (1)$$

In equation 1 d is the number of space-time dimensions, a is the lattice spacing, m_0 is the bare mass and λ_0 is the bare coupling. It is possible to relabel the quantities to obtain a simpler looking action as in equation 4. In order to achieve that, we first multiply and divide by a parameter κ, and also distribute a^2 inside the summation to obtain equation 2

$$S_E^l = \frac{a^{d-2}}{\kappa}\sum_x \left\{ \frac{\kappa}{2}(2d + (m_0a)^2)\phi_x^2 - \kappa\sum_\mu \phi_x\phi_{x+\mu} + \kappa a^2\frac{\lambda_0}{4!}\phi_x^4 \right\} \quad (2)$$

Now, we can define $\kappa = 1/(2d + (m_0a)^2)$ to leave ϕ_x^2 term alone, and absorb the overall multiplicative term in our field by defining $\chi_x = \sqrt{a^{d-2}/\kappa}\phi_x$, which results in equation 3.

$$S_E^l = \sum_x \left\{ \frac{1}{2}\chi_x^2 - \kappa\sum_\mu \chi_x\chi_{x+\mu} + \frac{\kappa^2a^2\frac{\lambda_0}{4!}}{a^{d-2}}\chi_x^4 \right\} \quad (3)$$

Finally by labeling $\kappa^2\lambda_0/4!a^{d-4}$ as g, we get equation 4 as below, which will be the form we will be using in our calculations.

$$S_E^l = \sum_x \left\{ \frac{1}{2}\chi_x^2 - \kappa\sum_\mu \chi_x\chi_{x+\mu} + g\chi_x^4 \right\} \quad (4)$$

With the MC we are going to develop, we are interested in calculating the following quantities shown in equations 5, 6, 7 and 8, where L is the number of lattice points in a single direction.

$$\sigma = \frac{1}{L^d}\sum_x \left\{ \langle\chi_x^2\rangle \right\} \quad (5)$$

$$X = \frac{1}{L^d}\sum_{x,y} \{\langle\chi_x\chi_y\rangle\} \quad (6)$$

$$F = \frac{1}{L^d}\sum_{x,y} \left\{ \langle\chi_x\chi_y\rangle cos(\frac{2\pi\Delta\tau}{L}) \right\} \quad (7)$$

$$M(L) = \frac{2sin(\pi/L)}{\sqrt{X/F - 1}} \quad (8)$$

In our MC calculation, our goal will be to obtain $M(L)$ for a given $(m_0a)^2$ and g and relate it to the physical mass of the scalar field, M_{phys}. Let's show that at large L, $M(L) = M_{phys}a$ relation holds true. Up to first order in momentum, pair correlation function is given as in equation 9.

$$\langle\phi(x)\phi(y)\rangle = \int d^dp \frac{Z}{p^2 + M_{phys}^2} e^{ip(x-y)} \quad (9)$$

We can replace summations in equations 6 and 7 with integrals appropriately (to keep dimension same, when upgrading lattice index x to the space-time point x we will need to divide by lattice spacing a) and put these integral definitions into equation 8 as can be seen in equation 10 below (We also converted χ_x back to $\phi(x)$.

$$M(L) = \frac{2sin(\pi/L)}{\sqrt{\frac{\frac{\int d^dxd^dy\langle\phi(x)\phi(y)\rangle a^{d-2}}{L^da^{2d}\kappa}}{\frac{\int d^dxd^dycos(\frac{2\pi/(x_0-y_0)}{La})\langle\phi(x)\phi(y)\rangle a^{d-2}}{L^da^{2d}\kappa}} - 1}} \quad (10)$$

Now, we can insert 9 into 10, do the obvious simplifications, and since integral only depend on $x - y$, we can change integration variables into x and $x - y$. Notice after this change integration over x terms will also cancel to give equation 11.

$$M(L)^2 = \frac{4sin^2(\pi/L)}{\frac{\int d^d(x-y)\int d^dp\frac{Z}{p^2+M_{phys}^2}e^{ip(x-y)}}{\int d^d(x-y)cos(\frac{2\pi/(x_0-y_0)}{La})\int d^dp\frac{Z}{p^2+M_{phys}^2}e^{ip(x-y)}} - 1} \quad (11)$$

By performing integrals over space-time first, we will obtain delta functions in momentum as in equation 12.

$$M(L)^2 = \frac{4sin^2(\pi/L)}{\frac{\int d^dp \frac{\delta^d(p)}{p^2+M_{phys}^2}}{\int d^dp \frac{\delta^{d-1}(p_{i\neq 0})\frac{\delta(p_0-\frac{2\pi}{La})+\delta(p_0+\frac{2\pi}{La})}{2}}{p^2+M_{phys}^2}} - 1} \tag{12}$$

Finally, when we apply the delta functions and use the fact that L is large ($2\pi/La$ is much larger than M_{phys}), we obtain equation 13.

$$M(L)^2 = \frac{4sin^2(\pi/L)}{\frac{1}{M_{phys}^2}} = \frac{(2\pi/L)^2}{(2\pi/LaM_{phys})^2} = (M_{phys}a)^2 \\ \frac{\frac{1}{M_{phys}^2}}{\frac{1}{(\frac{2\pi}{La})^2+M_{phys}^2}} - 1 \tag{13}$$

By taking the positive solution (Both $M(L)$ we will calculate and physical mass are positive quantities), we conclude for large enough L the relation 14 will be a good approximation.

$$M(L) = M_{phys}a \tag{14}$$

II. Calculation of Exact Results in Specific Cases

It is helpful to calculate exact results when possible in order to test the Monte Carlo algorithm. In the following subsections we consider two of special cases, which allow us to calculate exact results.

A. $\kappa = 0$ *Case*

This version is quite straightforward since no interaction between different space-time lattice points is present. The general expression for a two-point correlation function will be then as in equation 15.

$$\langle \chi_x \chi_y \rangle = \frac{\int_{-\infty}^{\infty} d\chi_x d\chi_y \chi_x \chi_y e^{-0.5\chi_x^2 - g\chi_x^4} e^{-0.5\chi_y^2 - g\chi_y^4}}{\int_{-\infty}^{\infty} d\chi_x e^{-0.5\chi_x^2 - g\chi_x^4} \int_{-\infty}^{\infty} d\chi_y e^{-0.5\chi_y^2 - g\chi_y^4}} \tag{15}$$

Notice that in equation 15, when $x \neq y$ the integrals will be separable resulting in two odd integrals over a symmetric interval, hence the pair-correlation function for $x \neq y$ yields 0. For $x = y$ case the integrals can be performed via Mathematica as a function of coupling g. Since all the lattice points will give the same result, σ in equation 5 reduces to $\langle \chi^2 \rangle$ (evaluated at any x). Similarly, all the non-diagonal entries of X and F in equations 6 and 7 will be zero, and they will be equal to σ.

B. $g = 0$ *Case*

This case corresponds to free scalar boson, which the pair correlation function is known to be as in equation 16, where M is the matrix that enables us to write the action in the form of 17.

$$\langle \chi_x \chi_y \rangle = M_{xy}^{-1} \tag{16}$$

$$S_E^l = \sum_x \left\{ \frac{1}{2}\chi_x^2 - \kappa \sum_\mu \chi_x \chi_{x+\mu} \right\} = \vec{\chi}^T M \vec{\chi} \tag{17}$$

Since we are working in a lattice and many entries are zero, M will be a finite and sparse matrix. For larger dimensions and lattice sizes computation of M^{-1} will be memory intensive, but a full computation of inverse is not needed. Notice σ is a scaled version of the trace of the inverse matrix, which can be computed by summing the multiplicative inverses of eigenvalues of M. X is simply the sum of all elements of the inverse matrix, which can be computed as in equation 18 where $\vec{1}$ is a vector of ones, and $\vec{s}$ can be obtained by solving the linear equation system 19.

$$X = \vec{1}^T (M^{-1}\vec{1}) = \vec{1}^T \vec{s} \tag{18}$$

$$M\vec{s} = \vec{1} \tag{19}$$

F can also be obtained in a similar way, one only needs to select the vectors that multiply the matrix from left and right more carefully, as in equation 20, where $sin(2\pi\vec{x}_0/L)$ refers to a vector of sines of time components of every lattice point. By using the same trick as in equation 19 these terms can be calculated as solutions of linear equation systems without inverting the matrix M.

$$F = sin(\frac{2\pi\vec{x}_0}{L})^T M^{-1} sin(\frac{2\pi\vec{y}_0}{L}) + cos(\frac{2\pi\vec{x}_0}{L})^T M^{-1} cos(\frac{2\pi\vec{y}_0}{L}) \tag{20}$$

This set of exact calculations were performed in MATLAB since it is optimized for matrix operations. The results of these exact calculations will be shown in the results section in comparison with results from our Monte Carlo.

III. The Algorithm

The Monte Carlo algorithm used can be explained in two main parts. First part is updating the field configurations in such a way that every configuration is sampled with probability $e^{-S_E^l(\vec{\chi})}$ causing lower action configurations will be dominant. This part can be further divided into two parts, regarding the update of sign of the field χ at every point and its magnitude. Second part is measuring

the observables given in equations 5, 6 and 7 given a configuration. In order to make the algorithm more efficient these measurements will be buried into the the sign update part of the code.

A. Sign Update

This part of the algorithm is using the single cluster methods developed for the spin $^1/_2$ Ising Model calculations. The sign of the field corresponds to the spin in the Ising model, while $\kappa|\chi_x\chi_y|$ corresponds to the $J\beta$ term in the Ising model. The difference is coupling can locally change in this code based on the absolute magnitude of the neighbouring lattice points, as opposed to global interaction term in the Ising model which is only a function of temperature. The properties of and reasoning behind this update was previously discussed in the report referenced in the item 2 of the Section VII, hence here only a summary of the method will be provided:

1) Select a random lattice site
2) Add neighbours with same sign to a cluster (a set connected lattice sites) with probability $1 - e^{-2\kappa|\chi_x\chi_y|}$
3) Repeat 2 for all lattice points added to cluster until no more lattice points are added and all existing sites in the cluster are exhausted
4) Flip the sign of the entire cluster (thus generate a different field configuration)
5) Use the properties of the cluster for measuring observables
6) Repeat steps above $O(^{L^d}/_{\langle Size_{Cluster}\rangle})$ times to update the whole lattice on average.

B. Regular Update

The sign update satisfies detailed balance condition, but not really ergodic since it can only change sign of the field at any lattice point. To address this point, another update based on the values of the field at a particular site and the neighbouring sites needs to be performed. This update will be described in terms of a single lattice point, but simply by looping through the entire volume, the values of the whole configuration can be modified.

Based on equation 4 and the fact that a configuration $\vec{\chi}$ has occurrence probability proportional with $e^{-S_E^L(\vec{\chi})}$, when field values of the neighbouring points were fixed the probability of field at a single point will be as in equation 21.

$$P(\chi_x) \propto e^{-(\chi_x - \alpha)^2/2} \times e^{-g\chi_x^4} \tag{21}$$

Where α is defined as in equation 22.

$$\alpha = \kappa \sum_{\mu} \chi_{x+\mu} \tag{22}$$

Notice that for the case when $g = 0$ this distribution is simply a Gaussian with mean α and standard deviation 1. In that case the current value of χ_x will not matter, and detailed balance equation (23) will be trivially satisfied since $P(i \to j)$ reduces to $P(j)$ and equation 24 is obtained. This makes sense, because we have the ability to generate the χ_x distribution we want directly, we do not need to rely on carefully tuned transition probabilities between different states.

$$P(i)P(i \to j) = P(j)P(j \to i) \tag{23}$$

$$P(i)P(j) = P(j)P(i) \tag{24}$$

It is clear we need a modification for the case $g \neq 0$. We will keep using the Gaussian distribution, but we will only accept the newly proposed value with a certain probability, which will be a function of old and proposed field values. We can use the detailed balance equation (23) to figure out that probability by labeling i as the current and j as the proposed state as in equation 25.

$$e^{-(\chi_{old} - \alpha)^2/2 - g\chi_{old}^4} e^{-(\chi_{new} - \alpha)^2/2} P_{acc} = e^{-(\chi_{new} - \alpha)^2/2 - g\chi_{new}^4} e^{-(\chi_{old} - \alpha)^2/2} \tag{25}$$

Then P_{acc} becomes as in equation 26, which will be the probability of acceptance of the newly proposed value in our MC algorithm.

$$P_{acc} = \frac{e^{-g\chi_{new}^4}}{e^{-g\chi_{old}^4}} \tag{26}$$

Since both of our updates satisfy detailed balance, and together they will be ergodic we have a valid Monte Carlo algorithm in which we visit different configurations proportional to their probabilities.

C. Observables

It is possible to calculate the observables given in equations 5, 6 and 7 very accurately with a double loop through the lattice (representing x and y summations). However, this method is clearly very inefficient. Instead we choose to measure the observables using the cluster obtained in the sign update section. The idea is that the cluster grown in the sign update is a good sample of points with the same sign, hence for X and F contributions outside the cluster will tend to average to zero

outside the cluster. Since the sign update is repeated $O(L^d/\langle Size_{Cluster}\rangle)$ times, by averaging that many measurements of X and F we can obtain a sample space comparable to the lattice volume. For example σ in equation 5 will become as in equation 27 for a given configuration and cluster. The expectation value will arise naturally when σ is averaged over many sign updates.

$$\sigma = \frac{1}{Size_{Cluster}} \sum_{x\in[C]} \chi_x^2 \tag{27}$$

One last trick, we can utilize that works for both F and X is that we do not have to perform a double loop within the cluster. We can only calculate correlation of every point within the lattice with a randomly selected point, since after many measurements we will get the same expectation value. This removes one of the summations with the $1/L^d$ term along with one of the summation symbols. We can use the first point we selected to grow the cluster from for this purpose, since that point will be randomly determined. Then based on our previous arguments, we can reduce the summation only within the cluster, and obtain equation 28. Same reasoning can be extended to F.

$$X = \langle \frac{1}{L^d} \sum_{y\in L^d} \chi_y \sum_{x\in L^d} \chi_x \rangle \rightarrow \langle \chi_r \sum_{x\in L^d} \chi_x \rangle \rightarrow \langle \chi_r \sum_{x\in[C]} \chi_x \rangle \tag{28}$$

IV. Implementation

We choose to implement single cluster algorithm in C++, mainly using standard libraries but making use of CERN ROOT libraries for random number generation[1] and plotting tools. For such MC simulations the performance of the code is critical, since low statistical errors are desired in a limited time.

A good place to start describing our code, is from its data structures. Below is the struct utilized to hold the value, sign, sine and cosine of time dimension and pointers to neighbouring lattice points, which is basically a multi-linked list structure. Such a structure will enable easy and elegant access to all neighbours and also make the processor read the address of neighbours preemptively[2]. Calculating and storing sine and cosine of the time dimension will be helpful, since this prevents calculation of the same quantity many times over [3] during the measurement of observable F.

```
struct LatticePts{ // Struct for a point in lattice, time is always 1st dimension
  bool fSign;
  double fVal;
  double trigTime[2]; // 0: cosine, 1: sine of (2*\pi*\timeDim)/LATLEN, cos(a-b)=cosa*cosb-sina*sinb
  LatticePts* nbrs[8] = {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL}; //pointers to neighbours
};
```

The script proceeds by creating a fixed size 2, 3 or 4 dimensional array of the LatticePts (tempLattice2D is a 2D array of LatticePts) objects and initializes all neighbours. An example for neighbour initialization for 1+1 dimension is given below, with special attention to periodic boundary conditions. This is the only part of the code, that requires dimension specific treatment, since in 2+1 and 3+1 dimensions, more neighbours need to be initialized. "latArr" vector in line 28 allows access to lattice without specifying dimensions for the rest of the code.

```
    for (unsigned int y = 0; y < INPUT.LATLEN; y++) {
      tempLattice2D[x][y].trigTime[0] = tempTrigTime[0];
      tempLattice2D[x][y].trigTime[1] = tempTrigTime[1];
      if(x == 0) {    //Special Attention if x is 0
        tempLattice2D[x][y].nbrs[0] = &tempLattice2D[INPUT.LATLEN-1][y];
        tempLattice2D[x][y].nbrs[1] = &tempLattice2D[x+1][y];
      }
      else if(x==(INPUT.LATLEN-1)) { // Special Attention if x=INPUT.LATLEN-1
        tempLattice2D[x][y].nbrs[0] = &tempLattice2D[x-1][y];
        tempLattice2D[x][y].nbrs[1] = &tempLattice2D[0][y];
      }
      else {//Default neighbours are spins at x+1 and x-1
        tempLattice2D[x][y].nbrs[0] = &tempLattice2D[x-1][y];
        tempLattice2D[x][y].nbrs[1] = &tempLattice2D[x+1][y];
      } // end of 1st dim (time dim)
      if (y==0){ //2nd dim
        tempLattice2D[x][y].nbrs[2] = &tempLattice2D[x][INPUT.LATLEN-1];
        tempLattice2D[x][y].nbrs[3] = &tempLattice2D[x][y+1];
      }
      else if(y==(INPUT.LATLEN-1)) {
        tempLattice2D[x][y].nbrs[2] = &tempLattice2D[x][y-1];
        tempLattice2D[x][y].nbrs[3] = &tempLattice2D[x][0];
      }
      else {
        tempLattice2D[x][y].nbrs[2] = &tempLattice2D[x][y-1];
        tempLattice2D[x][y].nbrs[3] = &tempLattice2D[x][y+1];
      } //end of 2nd dim
      latArr.push_back(&tempLattice2D[x][y]);
    } // end x (time)
  } // end y
```

The sign update and measurement of observables are performed in the following function. As the algorithm suggests we select a random site from

[1] Specifically TRandom3 with a period of $2^{19937}-1$ and 5 ns call time was chosen.

[2] Of course reading the field sign and value of the neighbours instead would be ideal for speed reasons, but that is much harder to implement, since those properties should be accessible and modifiable by 4 to 8 other lattice points.

[3] Critical because a trigonometric equation is combination of many summation and multiplication operations.

"latArr" [4], save its sign and value to temporary variables and flip its sign. This early flipping prevents checking this site again without the need of setting any flags. This results in less conditions to check and more cache space available for other purposes. We then push the initial point into a queue structure. A queue object is suitable here because we do not need to access a specific neighbour but to merely take the next one to analyze. In addition, the queue is automatically feeding it's front element to processor cache making it very efficient for our purposes. In the following while loop, we keep growing the cluster until the queue is empty, with the same method we did for the first site. Meanwhile, we estimate the observables via the values of the sites within the cluster.

```
void Lattice::signUpdate(QftInputPar INPUT, TRandom3& inRand) {
  LatticePts* curPts = latArr[inRand.Integer(INPUT.LATSIZE)]; //
      select a random site
  queue<LatticePts*> tempQ; // temporary queue to store lattice pts
      in queue
  double pNotBond; // probability of not adding a lattice point to
      queue
  bool tempSign = curPts->fSign; double tempVal = curPts->fVal; //
      save the current sign and value
  curPts->fSign = !tempSign; // flip the sign now to prevent visiting
       again
  tempQ.push(curPts); // push into the queue to add to cluster, and
      check neighbours

  double tempTrigTime[2] = {curPts->trigTime[0],curPts->trigTime[1]};
       // save current time trigs
  observable[1] = 0; // contribution to observables from the 1st
      point in cluster
  observable[2] = 0; observable[0] = 0;
  cSize = 0;
  while(!tempQ.empty()) { //Keep expanding the cluster until the
      queue is empty (nowhere to look)
    curPts = tempQ.front(); // Get the first element of the queue
    cSize++;
    observable[0] += pow(curPts->fVal,2);
    double tempMult = tempVal*curPts->fVal; // temporary storage,
      since will be used twice
    observable[1] += tempMult; // Calculate correlation observable
    observable[2] += tempMult*(tempTrigTime[0]*curPts->trigTime[0]+
        tempTrigTime[1]*curPts->trigTime[1]); // Calculate Time
      Weighted correlation observable
    tempQ.pop(); //Remove the element obtained from queue
    for(unsigned int nbrCtr=0; nbrCtr<INPUT.TWODIM; nbrCtr++) {
      if(curPts->nbrs[nbrCtr]->fSign == tempSign) { // If same sign (
      otherwise not added to cluster)
        pNotBond = Exp(-2*INPUT.KAPPA*curPts->fVal*curPts->nbrs[
      nbrCtr]->fVal); // calculate prob
        if(inRand.Rndm() > pNotBond) { //If prob is small enough
          curPts->nbrs[nbrCtr]->fSign = !tempSign; // flip sign now,
      to prevent revisiting
          tempQ.push(curPts->nbrs[nbrCtr]); // add to queue, hence to
       the cluster
        } // end of lucky rng
      } // end of field sign equality check
    } // end of neighbours of the current point loop
  } // end of queue emptying while loop

  observable[0]/=(double)cSize;
} // end of signUpdate function
```

The regular update function shown below, works in a straightforward manner; it loops over all lattice sites and updates every point based on its neighbours and a gaussian random variable. Finally, to preserve detailed balance when $g \neq 0$, the update is accepted or rejected based on the method described in the algorithm section.

```
void Lattice::valUpdate(QftInputPar INPUT, TRandom3& inRand) {
  //observable[0] = 0;
  for(unsigned int lCtr = 0; lCtr<INPUT.LATSIZE; lCtr++) { // loop
      over lattice positions
    double fValNew = 0; //init neighbour contribution
    LatticePts* curPts = latArr[lCtr];
    for(unsigned int nbrCtr=0;nbrCtr<INPUT.TWODIM;nbrCtr++) {// loop
      over neighbours and add them
      fValNew+=curPts->nbrs[nbrCtr]->fVal*(curPts->nbrs[nbrCtr]->
      fSign==curPts->fSign ? 1.0:-1.0);
    }
    fValNew*=INPUT.KAPPA; // multiply with KAPPA
    fValNew+=inRand.Gaus(0,1); // add normal rv
    double fValSqr[2] = {pow(curPts->fVal,2),pow(fValNew,2)}; // old
      and new field squares
    double pAccept = Exp(INPUT.COUPLING*(pow(fValSqr[0],2)-pow(
      fValSqr[1],2)));
    if(pAccept > inRand.Rndm()) {
      curPts->fVal = Abs(fValNew);
      curPts->fSign = (fValNew>0 ? curPts->fSign : !curPts->fSign);
      //observable[0] += fValSqr[1];
    }
    //else observable[0] += fValSqr[0];
  } // end of loop over lattice positions
} // end of valUpdate function
```

When main function calls sign and regular updates $O\left(L^d/\langle Size_{Cluster}\rangle\right)$ times, and average of the observables measured within that many updates are counted as a single measurement. This process is repeated many times, and statistics of the measurements were calculated.

The program is configured to run from command line with inputs d, L, κ or α (based on option selected), g, NSWEEP, version no, option code and optionally a previously saved field file (provides a start from equilibrium distribution). For running the program in an automated fashion for a set of input parameters bash scripts could be utilized. The results were then fitted with χ^2 method, again by using CERN ROOT libraries.

The code was not profiled specifically, but it was noted that running all the different parts of the project takes about 24 hours in total on a computer with 16GB of RAM and i7-6700HQ processors, with Ubuntu 16.04 as operating system.

V. Results

A. Comparison with Exact Results

Before moving onto the final results, we first check our Monte Carlo code by comparing its outputs with exact calculations when possible, as shown in Tables I and II. Both tables highlight very good agreement between MC and exact results.

In table II it is possible to see that $M(L) \approx M_{phys}a = m_0 a$ as expected since coupling g is set to zero. In table III, it is possible to see that $M(L)$ is independent of L in the case of $g = 0$.

[4]This is one part of the code where random memory access was not eliminated, fortunately this part does not limit the speed since it only happens once per cluster.

TABLE I: Comparison of exact and MC results in 1+1, 2+1 and 3+1 dimensions with $\kappa = 0$, $g = 0.1$, and L=16.

Type	σ	X	F
d=2 Exact	0.6155	0.6155	0.6155
d=2 MC	0.6152 ± 0.0004	0.612 ± 0.008	0.612 ± 0.008
d=3 Exact	0.6155	0.6155	0.6155
d=3 MC	0.6154 ± 0.0001	0.617 ± 0.007	0.617 ± 0.007
d=4 Exact	0.6155	0.6155	0.6155
d=4 MC	0.6155 ± 0.0001	0.614 ± 0.007	0.614 ± 0.007

TABLE II: Comparison of exact and MC results in 1+1, 2+1 and 3+1 dimensions with $(m_0 a)^2 = 0.25$ (fixes κ, but dimension dependent), $g = 0$, and L=16.

Type	σ	X	F	$M(L)$
d=2 Exact	1.6021	17.0000	10.5658	0.5000
d=2 MC	1.601 ± 0.001	17.00 ± 0.06	10.56 ± 0.03	0.500 ± 0.003
d=3 Exact	1.3198	25.0000	15.5380	0.5000
d=3 MC	1.3187 ± 0.0009	25.06 ± 0.08	15.57 ± 0.04	0.500 ± 0.003
d=4 Exact	1.1995	33.0000	20.5101	0.5000
d=4 MC	1.200 ± 0.002	33.3 ± 0.2	20.6 ± 0.1	0.497 ± 0.006

TABLE III: 1+1 dimensions with $g = 0$, dependence of $M(L)$ of L can be seen in this table, the dependence matches with the constant hypothesis.

L	$M(L)$
12	0.496 ± 0.003
16	0.500 ± 0.003
20	0.500 ± 0.003
24	0.500 ± 0.003

B. Project 1, Part 3: $(m_0 a)^2 = -1.5, L = 8, g = 0.1$

The results for this section were shown in Table IV, with errorbars less than 0.2% as required.

TABLE IV: MC Results for $(m_0 a)^2 = -1.5, L = 8, g = 0.1$ in 1+1,2+1 and 3+1 dimensions. Notice that all results have less than 0.2% error.

Dim	σ	X	F	$M(L)$
d=1+1	0.8296 ± 0.0003	6.02 ± 0.01	2.474 ± 0.002	0.639 ± 0.001
d=2+1	0.6857 ± 0.0001	2.9078 ± 0.0007	2.1085 ± 0.0004	1.2431 ± 0.0007
d=3+1	0.6561 ± 0.0001	2.3535 ± 0.0009	1.9416 ± 0.0006	1.662 ± 0.002

C. Computing the Lattice Spacing

We report the results in 1+1 dimensions with $g = 0.01$ and $(m_0 a)^2 = -0.5$, in table V. With $M(L)$ at $L = 48$ in table V and equation 14 it is possible to calculate the lattice spacing a for $M_{phys} = 1GeV$ as in equation 29.

$$a = \frac{M(L)}{M_{phys}} = 2.73 \times 10^{-4} MeV^{-1} \qquad (29)$$

TABLE V: 1+1 dimensions with $g = 0.01$ and $(m_0 a)^2 = -0.5$, dependence of $M(L)$ of L can be seen in this table, the dependence matches with the constant hypothesis especially for $L > 16$.

L	$M(L)$
8	0.298 ± 0.003
12	0.281 ± 0.002
16	0.275 ± 0.002
24	0.275 ± 0.002
36	0.274 ± 0.002
48	0.273 ± 0.002

D. Renormalization in 1+1 Dimensions

In this section we are studying how to tune (renormalize) m_0 by using the MC we have developed. We are studying 1+1 dimensions with $g = 0.01$ and $L = 48$, and testing whether the form shown in 30 holds true with the data we have taken for $\nu = 1$. In addition with a fit, we can compute f_0 and α_c as shown in Figure 1. Outputs of MC was shown explicitly in Table VI.

$$M(L) \approx M_{phys} a = f_0((m_0 a)^2 - \alpha_c)^\nu \qquad (30)$$

TABLE VI: 1+1 dimensions with $g = 0.01$ and $L = 48$, $(m_0 a)^2$ dependence of $M(L) \approx M_{phys} a$ can be seen in the table.

$(m_0 a)^2$	$M(L) \approx M_{phys} a$
-0.55	0.2018 ± 0.0005
-0.56	0.1864 ± 0.0005
-0.57	0.1700 ± 0.0004
-0.58	0.1535 ± 0.0003
-0.59	0.1363 ± 0.0003
-0.60	0.1184 ± 0.0002
-0.61	0.1002 ± 0.0002

Figure 1 shows that the form in equation 30 is a good fit to data, with a reduced chi^2 value around 1. The fit output for f_0 and α_c can also be seen in that figure.

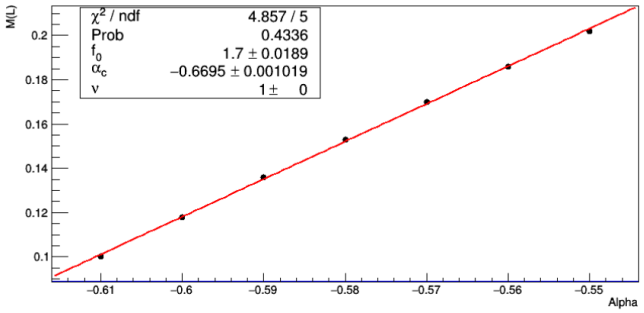

Fig. 1: $M(L) = M_{phys}a$ depends linearly on $(m_0a)^2$ for 1+1 dimensions and L=48. The reduced χ^2 value and the visual inspection of the data and fit verifies that hypothesis. Errorbars are plotted but they are smaller than the marker size. Fit values for f_0 and α_c are provided on the graph.

E. Renormalization in 2+1 Dimensions

Now we are studying 2+1 dimensions with $g = 0.01$ and $L = 48$, and testing whether the form shown in 30 holds true with the data we have taken for $\nu = 0.629971$. In addition with a fit, we can compute f_0 and α_c as shown in Figure 2. Outputs of MC was shown explicitly in Table VII.

TABLE VII: 2+1 dimensions with $g = 0.01$ and $L = 48$, $(m_0a)^2$ dependence of $M(L) \approx M_{phys}a$ can be seen in the table.

$(m_0a)^2$	$M(L) \approx M_{phys}a$
-0.68	0.200 ± 0.001
-0.69	0.178 ± 0.001
-0.70	0.1522 ± 0.0008
-0.71	0.1242 ± 0.0006
-0.72	0.0894 ± 0.0005

Figure 2 shows that the form in equation 30 is a good fit to data, with a reduced χ^2 value around 1. The fit output for f_0 and α_c can also be seen in that figure.

F. Renormalization in 3+1 Dimensions

Now we are studying 3+1 dimensions with $g = 0.01$ and $L = 48$, and testing whether the form shown in 30 holds true with the data we have taken for $\nu = 0.5$. In addition with a fit, we can compute f_0 and α_c as shown in Figure 3. Outputs of MC was shown explicitly in Table VIII.

Figure 3 shows that the form in equation 30 is a good fit with $\nu = 0.5$ to data, with a reduced chi^2

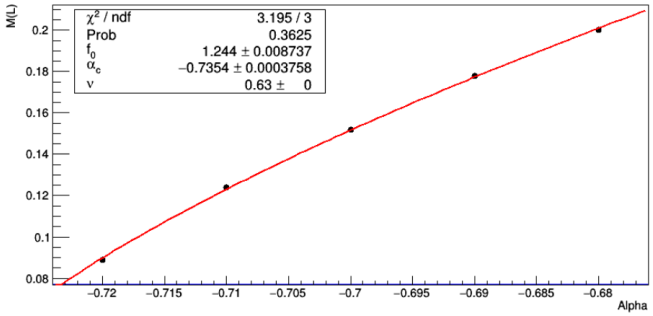

Fig. 2: $(m_0a)^2$ dependence of $M(L) = M_{phys}a$ for 2+1 dimensions and L=48 is well modeled with $\nu = 0.629971$. The reduced χ^2 value and the visual inspection of the data and fit verifies that hypothesis. Errorbars are plotted but they are smaller than the marker size. Fit values for f_0 and α_c are provided on the graph.

TABLE VIII: 3+1 dimensions with $g = 0.01$ and $L = 12$, $(m_0a)^2$ dependence of $M(L) \approx M_{phys}a$ can be seen in the table.

$(m_0a)^2$	$M(L) \approx M_{phys}a$
-0.55	0.578 ± 0.002
-0.60	0.531 ± 0.002
-0.65	0.478 ± 0.002
-0.70	0.421 ± 0.002
-075	0.354 ± 0.001

value around 1. The fit output for f_0 and α_c can also be seen in that figure.

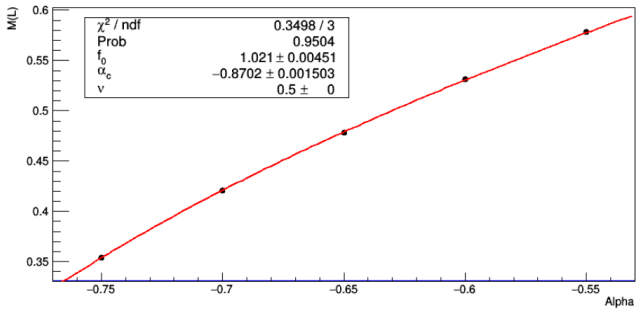

Fig. 3: $(m_0a)^2$ dependence of $M(L) = M_{phys}a$ for 3+1 dimensions and L=12 is well modeled with $\nu = 0.5$. The reduced χ^2 value and the visual inspection of the data and fit verifies that hypothesis. Errorbars are plotted but they are smaller than the marker size. Fit values for f_0 and α_c are provided on the graph.

G. Spontaneous Symmetry Breaking in 2+1 Dimensions

In this section we study the difference between L dependence of X at $(m_0 a)^2 = -0.7$ and $(m_0 a)^2 = -0.8$. The former is below the $\alpha_c = -0.7354$ found in the previous section and the latter is above, causing X to behave very different for two cases, as shown in Table IX, and Figures 4 and 5. As the fit in Figure 4 shows, for the case $(m_0 a)^2 = -0.8$, $M(L)$ increases cubically with L. This is because in this regime, the expectation value of the field at a single lattice point is no longer zero. Hence the pair-correlation between fields is centered around a non-zero value resulting in correlations even at the furthest lattice points. Therefore, in this regime X, sum of all correlations increase proportional to the volume of the lattice, as opposed to more or less constant version in the $(m_0 a)^2 = -0.7$ regime in which correlation functions decay after a certain length, hence increasing the lattice size has no large impact on X. Figures 4 and 5 support this reasoning.

TABLE IX: 2+1 dimensions with $g = 0.01$, L dependence of X is shown for $(m_0 a)^2 = -0.7 > \alpha_c$ and $(m_0 a)^2 = -0.8 < \alpha_c$.

L	X at $(m_0 a)^2 = -0.7$	X at $(m_0 a)^2 = -0.8$
16	195 ± 3	3150 ± 10
24	219 ± 2	10610 ± 20
32	225 ± 1	25030 ± 50
48	226.3 ± 0.8	84340 ± 60

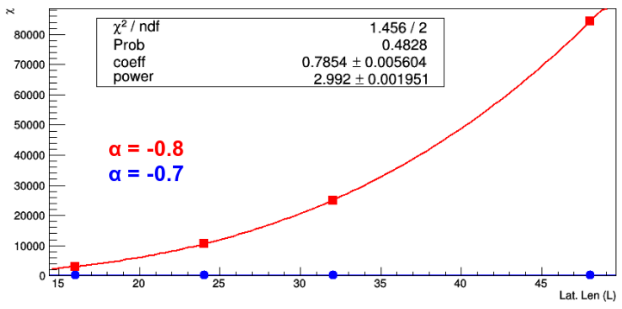

Fig. 4: Observable X as a function of lattice size in 2+1 dimesions for $(m_0 a)^2 = -0.7$ and $(m_0 a)^2 = -0.8$. In the second case, the vacuum expectation value is greater than zero, and correlation length goes to infinity, hence the X increases with L^d due to contributions from increasing lattice size.

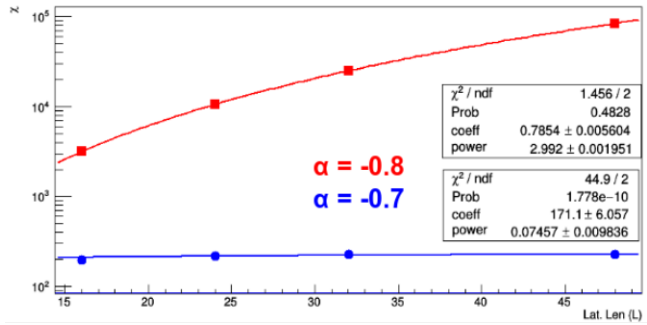

Fig. 5: Observable X as a function of lattice size in 2+1 dimesions for $(m_0 a)^2 = -0.7$ and $(m_0 a)^2 = -0.8$, y axis in log scale for resolving the details of the first case. In the first case, correlation length is finite and increasing the lattice size does not contribute to the sum of correlations since at large distances correlation decays.

VI. Summary and Conclusion

A Monte Carlo simulation for scalar bosons with quartic interaction on a periodic lattice in 2, 3 and 4 dimensions was developed. It was found out that working on a periodic lattice in Euclidean time is a possible alternative to perturbation theory we have developed in the class. Through this project, ideas of renormalization and spontaneous symmetry breaking are also observed. Beyond quantum field theory the project was helpful because it led to considerations such as using Markov chains in Monte Carlo, and coding faster running simulation software.

VII. Guides

In this project no real physics articles were read or studied, the information guided me are listed below in a non-professional but honest way:

1) Dr. Shailesh Chandrasekharan's online lectures and project description, <https://webhome.phy.duke.edu/~sch/763/>, Access Date: 10/2018
2) Baran Bodur, Monte Carlo Simulation of the Spin $^1/_2$ Ising Model in 3 Dimensions and Behavior of Spin Susceptibility Around the Critical Temperature for Ferromagnetic Transition
3) Occasional Wikipedia
4) <http://www.cplusplus.com/>
5) <https://root.cern.ch/>
6) <https://www.mathworks.com/products/matlab.html>