PHYS 516 Assignment 3
Molecular Dynamics Simulation **Cathie Tsz Yan SO**

1. (*Liouville's theorem*)

$$x'(x,p) = x + p\Delta + \frac{1}{2}a(x)\Delta^2$$

$$p'(x,p) = p + \frac{a(x) + a\left(x + p\Delta + \frac{1}{2}a(x)\Delta^2\right)}{2}\Delta$$

$$\left|\frac{\partial(x',p')}{\partial(x,p)}\right| = \frac{\partial x'}{\partial x}\frac{\partial p'}{\partial p} - \frac{\partial x'}{\partial p}\frac{\partial p'}{\partial x}$$

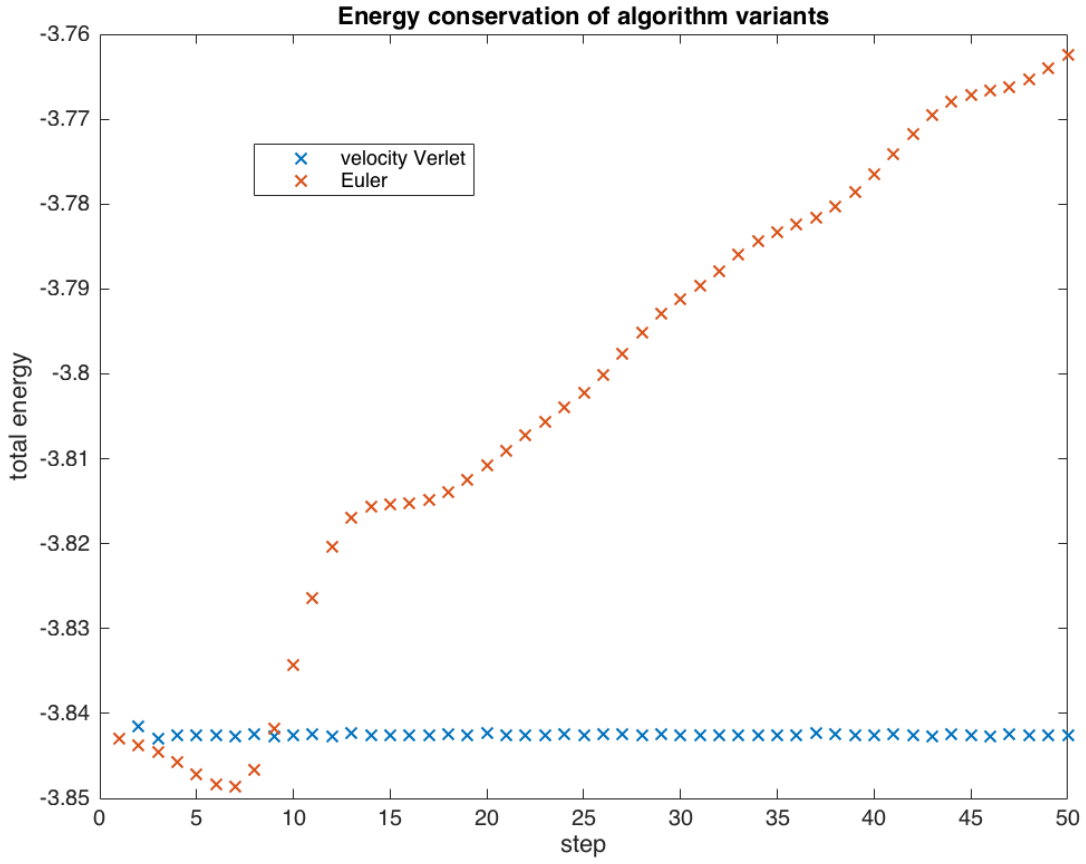
$$= \left(1 + \frac{\Delta^2}{2}\frac{\partial a(x)}{\partial x}\right)\left(1 + \frac{\Delta^2}{2}\frac{\partial a(x)}{\partial x}\right) - \Delta\left(\frac{\Delta}{2}\left(\frac{\partial a(x)}{\partial x} + \frac{\partial a(x)}{\partial x}\left(1 + \frac{\Delta^2}{2}\frac{\partial a(x)}{\partial x}\right)\right)\right)$$

$$= 1 + \Delta^2\frac{\partial a(x)}{\partial x} + \frac{\Delta^4}{4}\left(\frac{\partial a(x)}{\partial x}\right)^2 - \frac{\Delta^2}{2}\left(2\frac{\partial a(x)}{\partial x}\right) - \frac{\Delta^2}{2}\left(\frac{\Delta^2}{2}\frac{\partial a(x)}{\partial x}\right)$$

$$= 1$$

QED

2. (*Algorithm variants*)



The velocity Verlet algorithm obviously conserves the total energy better.

Modification of SingleStep() for Euler algorithm:

```
/*--------------------------------------------------------------------------*/
void SingleStep() {
/*--------------------------------------------------------------------------
  r & rv are propagated by DeltaT in time using the Euler method.
----------------------------------------------------------------------------*/
  int n,k;

 ComputeAccel();

 for (n=0; n<nAtom; n++)
     for (k=0; k<3; k++) {
         r[n][k] += DeltaT*(rv[n][k]+ra[n][k]*DeltaT/2.0);
         rv[n][k] += DeltaT*ra[n][k];
     };

  ApplyBoundaryCond();

}
```

3. (*Velocity autocorrelation*)
[Modifications are bold and underlined.]

(VAC.h)

```
/*****************************************************************************
File VAC.h is an include file for program VAC.c.
*****************************************************************************/
#define NMAX 100000  /* Maximum number of atoms which can be simulated */
#define RCUT 2.5     /* Potential cut-off length */
#define PI 3.141592653589793
/* Constants for the random number generator */
#define D2P31M 2147483647.0
#define DMUL 16807.0

/* modifications */
#define STEPCORR 2000
#define NSAMPLE 100
double v0[NMAX][3];
double VAC[STEPCORR];
void CAL_VAC();

/* Functions & function prototypes ******************************************/

double SignR(double v,double x) {if (x > 0) return v; else return -v;}
double Dmod(double a, double b) {
     int n;
     n = (int) (a/b);
     return (a - b*n);
}
double RandR(double *seed) {
     *seed = Dmod(*seed*DMUL,D2P31M);
     return (*seed/D2P31M);
}
void RandVec3(double *p, double *seed) {
     double x,y,s = 2.0;
     while (s > 1.0) {
       x = 2.0*RandR(seed) - 1.0; y = 2.0*RandR(seed) - 1.0; s = x*x + y*y;
     }
     p[2] = 1.0 - 2.0*s; s = 2.0*sqrt(1.0 - s); p[0] = s*x; p[1] = s*y;
}
void InitParams();
void InitConf();
void ComputeAccel();
void SingleStep();
void HalfKick();
void ApplyBoundaryCond();
void EvalProps();

/* Input parameters (read from an input file in this order) ******************/

int InitUcell[3];   /* Number of unit cells */
double Density;     /* Number density of atoms (in reduced unit) */
double InitTemp;    /* Starting temperature (in reduced unit) */
double DeltaT;      /* Size of a time step (in reduced unit) */
int StepLimit;      /* Number of time steps to be simulated */
int StepAvg;        /* Reporting interval for statistical data */

/* Constants ****************************************************************/
```

```
double Region[3];  /* MD box lengths */
double RegionH[3]; /* Half the box lengths */
double DeltaTH;    /* Half the time step */
double Uc, Duc;    /* Potential cut-off parameters */

/* Variables ************************************************************/

int nAtom;            /* Number of atoms */
double r[NMAX][3];    /* r[i][0|1|2] is the x|y|z coordinate of atom i */
double rv[NMAX][3];   /* Atomic velocities */
double ra[NMAX][3];   /* Acceleration on atoms */
double kinEnergy;     /* Kinetic energy */
double potEnergy;     /* Potential energy */
double totEnergy;     /* Total energy */
double temperature;   /* Current temperature */
int stepCount;        /* Current time step */
/************************************************************************/
```

(VAC.c)

```
#include <stdio.h>
#include <math.h>
#include "VAC.h"
#include <time.h>

int main(int argc, char **argv) {
    double cpu, cpu1, cpu2;
    InitParams();
    InitConf();

    /* modifications: add and initialize necessary variable declarations */
    double sumZt[STEPCORR], sum2Zt[STEPCORR], avgZt[STEPCORR], stdZt[STEPCORR], v;
    int outer, i, n, k;
    for (i=0; i<STEPCORR; i++) {
        sumZt[i]=0;
        sum2Zt[i]=0;
    }

    /* modifications: add outer loop */
    for (outer=0; outer<NSAMPLE; outer++) {
        for (n=0; n<nAtom; n++) {
            for (k=0; k<3; k++) {
                v0[n][k] = rv[n][k];
            }
        }

        ComputeAccel(); /* Computes initial accelerations */
        cpu1 = ((double) clock())/CLOCKS_PER_SEC;

        for (stepCount=1; stepCount<=StepLimit; stepCount++) {
            /* modifications: call function to calculate VAC */
            CAL_VAC(stepCount);

            SingleStep();
            if (stepCount%StepAvg == 0) EvalProps();
        }
        cpu2 = ((double) clock())/CLOCKS_PER_SEC;
        cpu = cpu2-cpu1;
        /* printf("%le %le %le\n",cpu1, cpu2, cpu); */

        /* modifications: normarlize, evaluate, and print VAC */
        for (i=0; i<StepLimit; i++) {
            v = VAC[i]/VAC[0];
            sumZt[i] += v;
            sum2Zt[i] += v*v;
        }

    }

    /* modificatins: calculate statistics of VAC and print */
    for (i=0; i<StepLimit; i++) {
        avgZt[i] = sumZt[i]/NSAMPLE;
        stdZt[i] = sqrt((sum2Zt[i]/NSAMPLE-avgZt[i]*avgZt[i])/(NSAMPLE-1));
        printf("Averaged Z(%i) = %f +- %e\n", i, avgZt[i],stdZt[i]);
    }
    return 0;
}

/*-------------------------------------------------------------------------*/
```

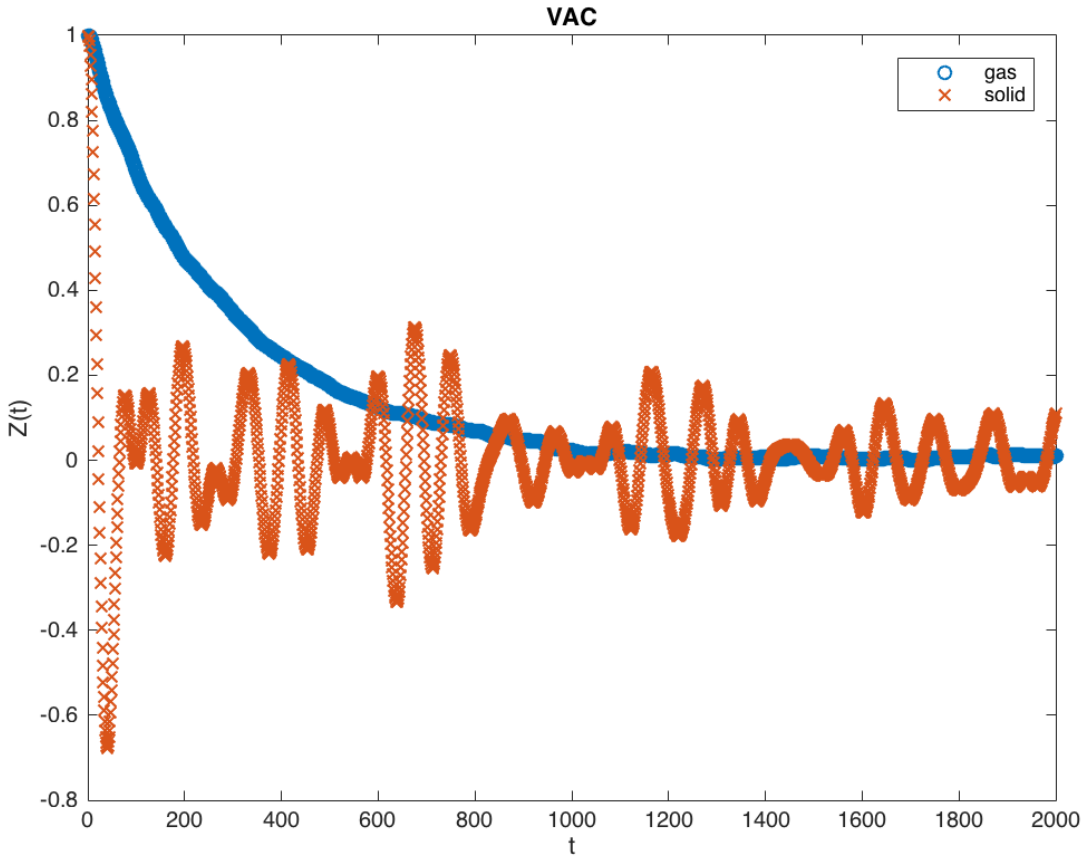
```
void CAL_VAC(int VACstep) {
    /*--------------------------------------------------------------------------------
     modifications: function to calculate VAC
     --------------------------------------------------------------------------------*/
    int n, k;
    for (n=0; n<nAtom; n++) {
        for (k=0; k<3; k++) {
            VAC[VACstep-1] += rv[n][k]*v0[n][k];
        }
    }
}

/*--------------------------------------------------------------------------------*/
void InitParams() {
    /*--------------------------------------------------------------------------------
     Initializes parameters.
     --------------------------------------------------------------------------------*/
    int k;
    double rr,ri2,ri6,r1;

    /* Reads control parameters */
    scanf("%d%d%d",&InitUcell[0],&InitUcell[1],&InitUcell[2]);
    scanf("%le",&Density);
    scanf("%le",&InitTemp);
    scanf("%le",&DeltaT);
    scanf("%d",&StepLimit);
    scanf("%d",&StepAvg);

    /* Computes basic parameters */
    DeltaTH = 0.5*DeltaT;
    for (k=0; k<3; k++) {
        Region[k] = InitUcell[k]/pow(Density/4.0,1.0/3.0);
        RegionH[k] = 0.5*Region[k];
    }

    /* Constants for potential truncation */
    rr = RCUT*RCUT; ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1=sqrt(rr);
    Uc = 4.0*ri6*(ri6 - 1.0);
    Duc = -48.0*ri6*(ri6 - 0.5)/r1;
}

/*--------------------------------------------------------------------------------*/
void InitConf() {
    /*--------------------------------------------------------------------------------
     r are initialized to face-centered cubic (fcc) lattice positions.
     rv are initialized with a random velocity corresponding to Temperature.
     --------------------------------------------------------------------------------*/
    double c[3],gap[3],e[3],vSum[3],vMag;
    int j,n,k,nX,nY,nZ;
    double seed;
    /* FCC atoms in the original unit cell */
    double origAtom[4][3] = {{0.0, 0.0, 0.0}, {0.0, 0.5, 0.5},
        {0.5, 0.0, 0.5}, {0.5, 0.5, 0.0}};

    /* Sets up a face-centered cubic (fcc) lattice */
    for (k=0; k<3; k++) gap[k] = Region[k]/InitUcell[k];
    nAtom = 0;
    for (nZ=0; nZ<InitUcell[2]; nZ++) {
        c[2] = nZ*gap[2];
        for (nY=0; nY<InitUcell[1]; nY++) {
            c[1] = nY*gap[1];
```

```
            for (nX=0; nX<InitUcell[0]; nX++) {
                c[0] = nX*gap[0];
                for (j=0; j<4; j++) {
                    for (k=0; k<3; k++)
                        r[nAtom][k] = c[k] + gap[k]*origAtom[j][k];
                    ++nAtom;
                }
            }
        }
    }

    /* Generates random velocities */
    seed = 13597.0;
    vMag = sqrt(3*InitTemp);
    for(k=0; k<3; k++) vSum[k] = 0.0;
    for(n=0; n<nAtom; n++) {
        RandVec3(e,&seed);
        for (k=0; k<3; k++) {
            rv[n][k] = vMag*e[k];
            vSum[k] = vSum[k] + rv[n][k];
        }
    }
    /* Makes the total momentum zero */
    for (k=0; k<3; k++) vSum[k] = vSum[k]/nAtom;
    for (n=0; n<nAtom; n++) for(k=0; k<3; k++) rv[n][k] = rv[n][k] - vSum[k];
}

/*----------------------------------------------------------------------------*/
void ComputeAccel() {
    /*------------------------------------------------------------------------
     Acceleration, ra, are computed as a function of atomic coordinates, r,
     using the Lennard-Jones potential.  The sum of atomic potential energies,
     potEnergy, is also computed.
     ------------------------------------------------------------------------*/
    double dr[3],f,fcVal,rrCut,rr,ri2,ri6,r1;
    int j1,j2,n,k;

    rrCut = RCUT*RCUT;
    for (n=0; n<nAtom; n++) for (k=0; k<3; k++) ra[n][k] = 0.0;
    potEnergy = 0.0;

    /* Doubly-nested loop over atomic pairs */
    for (j1=0; j1<nAtom-1; j1++) {
        for (j2=j1+1; j2<nAtom; j2++) {
            /* Computes the squared atomic distance */
            for (rr=0.0, k=0; k<3; k++) {
                dr[k] = r[j1][k] - r[j2][k];
                /* Chooses the nearest image */
                dr[k] = dr[k] - SignR(RegionH[k],dr[k]-RegionH[k])
                - SignR(RegionH[k],dr[k]+RegionH[k]);
                rr = rr + dr[k]*dr[k];
            }
            /* Computes acceleration & potential within the cut-off distance */
            if (rr < rrCut) {
                ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
                fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
                for (k=0; k<3; k++) {
                    f = fcVal*dr[k];
                    ra[j1][k] = ra[j1][k] + f;
                    ra[j2][k] = ra[j2][k] - f;
                }
```

```
                potEnergy = potEnergy + 4.0*ri6*(ri6-1.0) - Uc - Duc*(r1-RCUT);
            }
        }
    }
}

/*----------------------------------------------------------------------------*/
void SingleStep() {
    /*-------------------------------------------------------------------------
     r & rv are propagated by DeltaT in time using the velocity-Verlet method.
     -------------------------------------------------------------------------*/
    int n,k;

    HalfKick(); /* First half kick to obtain v(t+Dt/2) */
    for (n=0; n<nAtom; n++) /* Update atomic coordinates to r(t+Dt) */
        for (k=0; k<3; k++) r[n][k] = r[n][k] + DeltaT*rv[n][k];
    ApplyBoundaryCond();
    ComputeAccel(); /* Computes new accelerations, a(t+Dt) */
    HalfKick(); /* Second half kick to obtain v(t+Dt) */
}

/*----------------------------------------------------------------------------*/
void HalfKick() {
    /*-------------------------------------------------------------------------
     Accelerates atomic velocities, rv, by half the time step.
     -------------------------------------------------------------------------*/
    int n,k;
    for (n=0; n<nAtom; n++)
        for (k=0; k<3; k++) rv[n][k] = rv[n][k] + DeltaTH*ra[n][k];
}

/*----------------------------------------------------------------------------*/
void ApplyBoundaryCond() {
    /*-------------------------------------------------------------------------
     Applies periodic boundary conditions to atomic coordinates.
     -------------------------------------------------------------------------*/
    int n,k;
    for (n=0; n<nAtom; n++)
        for (k=0; k<3; k++)
            r[n][k] = r[n][k] - SignR(RegionH[k],r[n][k])
            - SignR(RegionH[k],r[n][k]-Region[k]);
}

/*----------------------------------------------------------------------------*/
void EvalProps() {
    /*-------------------------------------------------------------------------
     Evaluates physical properties: kinetic, potential & total energies.
     -------------------------------------------------------------------------*/
    double vSum,vv;
    int n,k;

    vSum = kinEnergy = 0.0;
    for (n=0; n<nAtom; n++) {
        vv = 0.0;
        for (k=0; k<3; k++) {
            vSum = vSum + rv[n][k];
            vv = vv + rv[n][k]*rv[n][k];
        }
        kinEnergy = kinEnergy + vv;
    }
    kinEnergy *= (0.5/nAtom);
```

```
    potEnergy /= nAtom;
    totEnergy = kinEnergy + potEnergy;
    temperature = kinEnergy*2.0/3.0;

    /* Print the computed properties */
    /* printf("%9.6f %9.6f %9.6f %9.6f\n",
           stepCount*DeltaT,temperature,potEnergy,totEnergy); */
}
```

Plot:



4. (*Split-operator formalism*)

$$iL = \{\cdots, H\} = \sum_{j=1}^{f}\left[\dot{x}_j \frac{\partial}{\partial x_j} + F_j \frac{\partial}{\partial p_j}\right] \quad (2.1)$$

where $f = 3N, H = \sum_{j=1}^{f} \frac{p_j^2}{2m} + V(x_1, \ldots, x_f), \dot{x}_j = \frac{\partial H}{\partial p_j} = \frac{p_j}{m}, \dot{p}_j = -\frac{\partial H}{\partial x_j} = -\frac{\partial V}{\partial x_j} = F_j$

Let f be any arbitrary function of x_i and p_i.

$$\dot{f}(x_j, p_j) = \sum_{j=1}^{f}\left(\dot{x}_j \frac{\partial}{\partial x_j} + \dot{p}_j \frac{\partial}{\partial p_j}\right) f = \sum_{j=1}^{f}\left(\frac{\partial H}{\partial p_j}\frac{\partial}{\partial x_j} - \frac{\partial H}{\partial x_j}\frac{\partial}{\partial p_j}\right) f = iLf$$

Consider $\Gamma = (x_j, p_j) = (x_1, \ldots, x_f, p_1, .., p_f)$ a point in phase-space.

$$\frac{d}{dt}\Gamma(t) = iL\Gamma(t)$$

$$U(t) = e^{iLt} \quad (2.2)$$

$$\Gamma(t) = U(t)\Gamma(0) \quad (2.3)$$

Proof:

$$\frac{d}{dt}\Gamma(t) = \frac{d}{dt}U(t)\Gamma(0) = \left(\frac{d}{dt}e^{iLt}\right)\Gamma(0) = \left(\frac{d}{dt}\sum_{n=0}^{\infty}\frac{(iLt)^n}{n!}\right)\Gamma(0)$$

$$= iL\left(\sum_{n=0}^{\infty}\frac{n(iLt)^{n-1}}{n!}\right)\Gamma(0) = iL\left(\sum_{n=0}^{\infty}\frac{(iLt)^n}{n!}\right)\Gamma(0) = iLU(t)\Gamma(0)$$

$$= iL\Gamma(t)$$

$$\Gamma(0) = U(0)\Gamma(0) = e^{iL(0)}\Gamma(0) = \Gamma(0)$$

$$iL = iL_1 + iL_2 \quad (2.4)$$

$$e^{i(L_1+L_2)t} = \left[e^{\frac{i(L_1+L_2)t}{P}}\right]^P = \left[e^{iL_1\left(\frac{\Delta t}{2}\right)}e^{iL_2\Delta t}e^{iL_1\left(\frac{\Delta t}{2}\right)}\right]^P + O\left(\frac{t^3}{P^2}\right) \quad (2.5)$$

where $\Delta t = \frac{t}{P}$

Proof of $exp(L\Delta) = exp\left(\frac{\Delta}{2}L_1\right) exp(\Delta L_2)\, exp\left(\frac{\Delta}{2}L_1\right) + O(\Delta^3)$:

$$LHS = 1 + (L_1 + L_2)\Delta + \frac{\Delta^2}{2}(L_1^2 + L_1L_2 + L_2L_1 + L_2^2) + O(\Delta^3)$$

$$RHS = \left(1 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{8}L_1^2\right)\left(1 + \Delta L_2 + \frac{\Delta^2}{2}L_2^2\right)\left(1 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{8}L_1^2\right) + O(\Delta^3)$$

$$= \left(1 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{8}L_1^2 + \Delta L_2 + \frac{\Delta^2}{2}L_1L_2 + \frac{\Delta^2}{2}L_2^2\right)\left(1 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{8}L_1^2\right) + O(\Delta^3)$$

$$= 1 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{8}L_1^2 + \Delta L_2 + \frac{\Delta^2}{2}L_1L_2 + \frac{\Delta^2}{2}L_2^2 + \frac{\Delta}{2}L_1 + \frac{\Delta^2}{4}L_1^2 + \frac{\Delta^2}{2}L_2L_1 + \frac{\Delta^2}{8}L_1^2 + O(\Delta^3)$$

$$= 1 + \Delta(L_1 + L_2) + \frac{\Delta^2}{2}(L_1^2 + L_1L_2 + L_2L_1 + L_2^2) + O(\Delta^3) = LHS$$

$$G(\Delta t) = U_1\left(\frac{\Delta t}{2}\right)U_2(\Delta t)U_1\left(\frac{\Delta t}{2}\right) = e^{iL_1\left(\frac{\Delta t}{2}\right)}e^{iL_2\Delta t}e^{iL_1\left(\frac{\Delta t}{2}\right)} \quad (2.6)$$

$$\Gamma_1[\Delta t; \Gamma(0)] = U_1(\Delta t)\Gamma(0) \quad (2.7)$$

$$\Gamma_2[\Delta t; \Gamma(0)] = U_2(\Delta t)\Gamma(0) \quad (2.8)$$

$$\Gamma(\Delta t) = U_1\left(\frac{\Delta t}{2}\right)U_2(\Delta t)U_1\left(\frac{\Delta t}{2}\right)\Gamma(0) = U_1\left(\frac{\Delta t}{2}\right)U_2(\Delta t)\Gamma_1\left[\frac{\Delta t}{2}; \Gamma(0)\right] \quad (2.9)$$

$$\Gamma(\Delta t) = U_1\left(\frac{\Delta t}{2}\right)\Gamma_2\left\{\Delta t; \Gamma_1\left[\frac{\Delta t}{2}; \Gamma(0)\right]\right\} \quad (2.10)$$

$$\Gamma(\Delta t) = \Gamma_1\left(\frac{\Delta t}{2}; \Gamma_2\left\{\Delta t; \Gamma_1\left[\frac{\Delta t}{2}; \Gamma(0)\right]\right\}\right) \quad (2.11)$$

$$e^{i(L_1+L_2)\Delta t} \approx e^{iL_1\left(\frac{\Delta t}{2}\right)}e^{iL_2\left(\frac{\Delta t}{2}\right)}e^{-iC\left(\frac{\Delta t^3}{24}\right)}e^{iL_2\left(\frac{\Delta t}{2}\right)}e^{iL_1\left(\frac{\Delta t}{2}\right)} \quad (2.12)$$

where $-iC = \left[(iL_1 + 2iL_2), [iL_1, iL_2]\right]$ (2.13)

$$iL_2 = \dot{x}\frac{\partial}{\partial x}; iL_1 = F(x)\frac{\partial}{\partial p} \quad (2.14)$$

$$G(\Delta t) = e^{\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}}e^{\Delta t\,\dot{x}\frac{\partial}{\partial x}}e^{\left(\frac{\Delta t}{2}\right)F(x)\frac{\partial}{\partial p}} \quad (2.15)$$

$$e^{c\frac{\partial}{\partial q}}f(q) = f(q + c) \quad (2.16)$$

Proof:

$$e^{c\frac{\partial}{\partial q}}f(q) = \sum_{n=0}^{\infty}\frac{f^{(n)}(0)}{n!}\sum_{k=0}^{\infty}\frac{c^k}{k!}\frac{\partial^k}{\partial q^k}q^n = \sum_{n=0}^{\infty}\frac{f^{(n)}(0)}{n!}\sum_{k=0}^{\infty}\frac{c^k}{k!}\frac{n!}{(n-k)!}q^{n-k}$$

$$= \sum_{n=0}^{\infty}\frac{f^{(n)}(0)}{n!}(q + c)^k = f(q + c)$$

by binomial theorem.

From (2.14),

$$\{\dot{x}, \dot{p}\} = \exp(iL\Delta t)\{x, p\} = e^{iL_1\left(\frac{\Delta t}{2}\right)}e^{iL_2\Delta t}e^{iL_1\left(\frac{\Delta t}{2}\right)}\{x, p\}$$

$$= \exp\left(\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}\right)\exp\left(\Delta\mathrm{t}\,\dot{x}\frac{\partial}{\partial x}\right)\exp\left(\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}\right)\{\mathrm{x}, \mathrm{p}\}$$

$$= \exp\left(\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}\right)\exp\left(\Delta\mathrm{t}\,\dot{x}\frac{\partial}{\partial x}\right)\{x, p + \frac{\Delta t}{2}F(x)\}$$

$$= \exp\left(\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}\right)\{x + \Delta\mathrm{t}\,\dot{x}, p + \frac{\Delta t}{2}F(x + \Delta\mathrm{t}\,\dot{x})\}$$

$$= \exp\left(\frac{\Delta t}{2}F(x)\frac{\partial}{\partial p}\right)\{x + \Delta\mathrm{t}\frac{p}{m}, p + \frac{\Delta t}{2}F\left(x + \Delta\mathrm{t}\frac{p}{m}\right)\}$$

$$= \left\{x + \frac{\Delta t\left(p + \frac{\Delta t}{2}F(x)\right)}{m}, p + \frac{\Delta t}{2}F(x) + \frac{\Delta t}{2}F\left(x + \Delta\mathrm{t}\frac{\left(p + \frac{\Delta t}{2}F(x)\right)}{m}\right)\right\}$$

$$= \left\{ x(0) + \Delta t\, \dot{x}(0) + \frac{\Delta t^2}{2}\frac{F[x(0)]}{m}, p + \frac{\Delta t}{2}(F[x(0)] + F[x(\Delta t)]) \right\}$$

Therefore with $\dot{x} = p/m$:

$$x(\Delta t) = x(0) + \Delta t\, \dot{x}(0) + \frac{(\Delta t)^2}{2}\frac{F[x(0)]}{m}$$

$$\dot{x}(\Delta t) = \dot{x}(0) + \frac{\Delta t}{2m}\{F[x(0)] + F[x(\Delta t)]\} \quad (2.17)$$

$$\dot{x}\left(\frac{\Delta t}{2}\right) = \dot{x}(0) + \frac{\Delta t}{2m}F[x(0)]$$

$$x(\Delta t) = x(0) + \Delta t\, \dot{x}\left(\frac{\Delta t}{2}\right)$$

$$\dot{x}(\Delta t) = \dot{x}\left(\frac{\Delta t}{2}\right) + \frac{\Delta t}{2m}F[x(\Delta t)] \quad (2.18)$$