```python
##########################################################################
###This is the main module which consists of all algorithms.
###Can be imported as package into other scripts.
### - Discretization of the Wiener process
### - Discretization of the time-interval
### - Refinement algorithm for the Wiener process (see thesis for explanation)
### - Numerical methods: Euler-Maruyama, Milstein and Wagner-Platen
###Author: Amr Umeri
###NumericalSDE.py
###Python 2.7
##########################################################################

from math import sqrt
import numpy as np
from scipy.stats import norm

#Global constant T. Can be changed
T=1
#Generates the increments for the Wiener process starting with 0
#n equals the amount of discetization points excluding 0
def wiener(n):
    dt = float(T)/n
    # generate a sample of n numbers from a
    # normal distribution and insert 0 as starting value
    rval = np.insert(norm.rvs(size=n, scale=sqrt(dt)),0,0)

    # This computes the Wiener process by forming the cumulative sum of
    # the random samples and returns its values
    return np.cumsum(rval)

#returns a time-grid of [0, T] with n+1 (including 0) discretization points
def timegrid(n):
    return np.linspace(0.0, T, n+1)

#returns a finer version of a given Wiener process
#see thesis for explanations
def refineWiener(a):
    n=a.size-1
    dt=float(T)/n

    rval=np.empty(a.size*2-1)

    for k in range((a.size-1)):
        rval[2*k] = a[k]
        rval[2*k+1] = norm.rvs(size=1, loc=(a[k]+a[k+1])/2, scale=sqrt(dt/4))
        ##prove it

    rval[a.size*2-2] = a[a.size-1]

    return rval

#Euler-scheme
def sde_euler(x0, a, b, w):
    n = w.size-1
    dt = float(T)/n
    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])

        Xval[k+1] = Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
    return Xval

#Milstein-scheme
def sde_milstein(x0, a, b, b_dv, w):
    n = w.size-1
```

```
    dt = float(T)/n
    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])
        b_dv_val        =   b_dv(Xval[k])

        Xval[k+1] = (Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
                    + float(1)/2*b_val*b_dv_val*((w[k+1]-w[k])**2-dt))
    return Xval



#Wagner-Platen-scheme
def sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w):
    n = w.size-1
    dt = float(T)/n

    Z = np.zeros(n)
    for k in range(0,n):
        Z[k] = 0.5*dt**1.5*((w[k+1]-w[k])/sqrt(dt) + norm.rvs()/sqrt(3))

    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])
        a_dv_val        =   a_dv(Xval[k])
        b_dv_val        =   b_dv(Xval[k])
        a_dvdv_val      =   a_dvdv(Xval[k])
        b_dvdv_val      =   b_dvdv(Xval[k])

        Xval[k+1] = (Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
                    + float(1)/2*b_val*b_dv_val*((w[k+1]-w[k])**2-dt)
                    + a_dv_val*b_val*Z[k]
                    + float(1)/2*(a_val*a_dv_val + float(1)/2*b_val**2*a_dvdv_val)*dt**2
                    + (a_val*b_dv_val +
                    float(1)/2*b_val**2*b_dvdv_val)*((w[k+1]-w[k])*dt-Z[k])
                    + float(1)/2*b_val*(b_val*b_dvdv_val +
                    (b_dv_val)**2)*(float(1)/3*(w[k+1]-w[k])**2-dt)*(w[k+1]-w[k]))
    return Xval
```