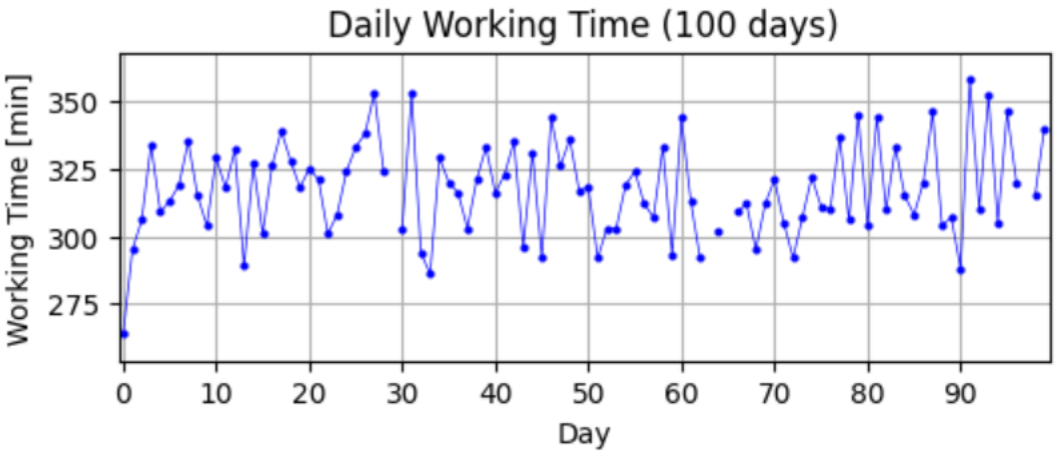
Step 6 Summary

The aim of step 6 is to develop the simulation study. This requires to run a number of larger simulations. As this takes quite a while, the first step is to save the simulation results using the pickle format. This allows us later to reload the Recorder instance of a simulation run and to drill deeper into plots or statistics without the need of a rerun of the simulation.

The first long simulation runs show a strange effect: the working time per day is growing beyond limits. Originally there was a soft working time limit of 3 hours postulated, i.e. the normal parcel delivery time was 18:00-21:00. Minor extensions were accepted. But now it turns out that the delivery time is going past 22:00, which is not acceptable for the customers.



Consequence: A daily working time limit has to be imposed. This means a change to the requirements. And this requires a change to the code.

Change Requirements (Specification):

The daily working time for the driver is intended to be limited to 3 hours. This needs to be taken into account in the route planning. Should the actual delivery process takes longer than originally planned, the driver is required to stop the delivery after 3 hours 20 minutes and return to the delivery centre with some of the remaining parcels still undelivered.

This implies a change to the basic code in class Driver and DeliveryCentre, and hence we have to rerun Step 4 Verification.

Changes to Class Driver

```
def process(self):
    yield self.rec.env.timeout(nextHour(self.rec.env, 18))
    while day(self.rec.env.now)<self.rec.days:
        self.arriveForWork()

        ## chnge to deal with time limit
        startTime = self.rec.env.now

        tour, parcels, addresses = self.DC.sendForDelivery()
        if len(parcels)==0:
            self.rec.trace("Nothing to do today")
            self.rec.recordTourLength(0)
        else:
            yield self.rec.env.timeout(PREP_TIME_PER_PARCEL*len(parcels))
            self.rec.recordTourLength(pathLength(tour))
            self.leaveForDelivery(tour, parcels, addresses)
            while len(self.parcels)>0:

                ## change to deal with time limit
                currentTime = self.rec.env.now
                if currentTime-startTime>=self.DC.timeLimit:
                    self.rec.trace("Timelimit reached")
                    while len(self.parcels)>0:
                        self.returns += [self.parcels[0]]
                        self.parcels = self.parcels[1:]
                    break

                # drive to customer
                custLocation = self.parcels[0].destination()
                cust = self.parcels[0].cust
                self.rec.trace("Driver drives to "+str(cust))
                yield from self.__drive(custLocation)
                self.rec.trace("Driver arrived at "+str(cust))
                # call at customer
                yield from cust.answerDoor()

                if cust.answersDoor:
                    while len(self.parcels)>0 and \
                            custLocation == self.parcels[0].destination():
                        cust.acceptParcel(self.parcels[0])
                        yield self.rec.env.timeout(random.expovariate(1/10))
                        self.parcels = self.parcels[1:]
                    cust.signOff()
                    yield self.rec.env.timeout(random.expovariate(1/10))
                else:
                    while len(self.parcels)>0 and  \
                            custLocation == self.parcels[0].destination():
                        self.returns += [self.parcels[0]]
                        self.parcels = self.parcels[1:]

            # return to delivery centre
            self.rec.trace("Driver returns to delivery centre")
            yield from self.__drive(self.DC.W)
            self.rec.trace("Driver arrived at delivery centre")

            for parcel in self.returns:
                self.DC.returnFromDelivery(parcel)
                yield self.rec.env.timeout(RETURN_TIME_PER_PARCEL)

        self.rec.recordParcelsLeftOver(len(self.DC.parcels)+len(self.DC.leftOver))

        yield self.rec.env.timeout(600)

        self.goesHome()

        yield self.rec.env.timeout(nextHour(self.rec.env, 18))
```

Changes to Class DeliveryCentre

```
class DeliveryCentre:

    def __init__(self, rec, M, W, limit, timeLimit):
        self.rec = rec
        self.M = M
        self.W = W
        self.limit = limit

        self.timeLimit = timeLimit

        self.leftOver = []    # list of parcels
        self.parcels = []     # list of parcels scheduled for delivery
        self.dest = []        # list of unique customer destinations
        self.tour = [self.W]  # tour planned for delivery

    def __accept(self, parcel):
        custLoc = parcel.destination()

        ## chnge to deal with time limit estimate
        timeEstimate = \
            len(self.parcels)*(PREP_TIME_PER_PARCEL + AVG_TIME_HANDOVER) \
            + len(self.dest)*(AVG_TIME_ANSWER_DOOR + AVG_TIME_SIGNOFF)

        if custLoc not in self.dest:
            MT = addTargets(self.M, self.dest + [custLoc])
            SH = createLoopG(MT, [self.W] + self.dest + [custLoc])

            ## chnge to deal with time limit estimate
            if pathLength(SH)<self.limit and \
                timeEstimate + pathLength(SH)/AVG_SPEED + \
                PREP_TIME_PER_PARCEL + AVG_TIME_ANSWER_DOOR + \
                AVG_TIME_HANDOVER + AVG_TIME_SIGNOFF <= self.timeLimit:

                self.parcels.append(parcel)
                self.dest += [custLoc]
                self.tour = SH
            else:
                self.leftOver.append(parcel)

        ## chnge to deal with time limit estimate
        elif timeEstimate + pathLength(self.tour)/AVG_SPEED + \
                PREP_TIME_PER_PARCEL + AVG_TIME_HANDOVER <= self.timeLimit:
            self.parcels.append(parcel)
        else:
            self.leftOver.append(parcel)
```