Matlab-Envelope (MENV) Documentation

Kiersten Ruisard

June 1, 2018

Contents

1 Introduction

MENV can be run from the command line ("clmenv.m") or GUI ("menv.m"). The GUI and original MENV functions were written by Hui Li. Kiersten added the command line functionality, and copied over much of Hui Li's original code into `clmenv`.

1.1 Envelope integrator

MENV integrates the KV envelope equations in X and Y:

$$X'' = -K_X X + \frac{2K}{X+Y} + \frac{E_X^2}{X^3} \tag{1}$$

The integration is handled in the Matlab function "menv_integrator," which integrates through a specified beam-line given focusing gradients. The integration uses the leap-frog method. The two sub-functions which form each integration step are given below:

```
function [x2,y2,xp2,yp2,d2,dp2] = step(x1,y1,xp1,yp1,d1,dp1)
  % function for single step
  x2 = x1+xp1*ds;
  y2 = y1+yp1*ds;
  d2 = d1+dp1*ds;
  [xpp,ypp,dpp] = calc_prim2(x2,y2,d2);
  xp2 = xp1+xpp*ds;
  yp2 = yp1+ypp*ds;
  dp2 = dp1+dpp*ds;
end
```

```
function [xpp1,ypp1,dpp1] = calc_prim2(x1,y1,d1)
  % function to calculate velocity impulse
  % Reiser eq. 4.191, 4.192
  xpp1 = -( kx*x1-2*K/(x1+y1)-Ex^2/(x1^3) );
  ypp1 = -( ky*y1-2*K/(x1+y1)-Ey^2/(y1^3) );
  dpp1 = irho - kx*d1;
end
```

1.2 Edge Focusing

Special consideration of dipole edge-focusing has to be made to accurately model UMER dynamics. In MENV, the edge focusing term is included as a hard-edged quadrupole with the same length as the dipole. The strength is not set automatically (ie, by setting a bend-angle), but an appropriate number must be determined by the user.

MENV allows for an asymmetry to be built into the horizontal and vertical edge-focusing, as separate gradient arrays are stored for each plane (`KX` and `KY`). The vertical focusing strength is set to the value chosen by the user for the quadrupole gradient strength at the dipole, while the horizontal focusing is defined as a scalar times this value. By default, the multiplicative factor is set to 0. No net horizontal focusing will be applied, following the argument that geometric focusing perfectly cancels the edge-defocusing in a sector dipole.

If you want to include the additional horizontal focusing observed in UMER dipoles due to sextupole terms (Kishek, R., & Cornacchia, M. (2010). Tech. Note, Modeling of UMER Dipoles.), you will need to change the scaling parameter `Dffx` that appears around line 30 in "menv_makekappaarray.m".

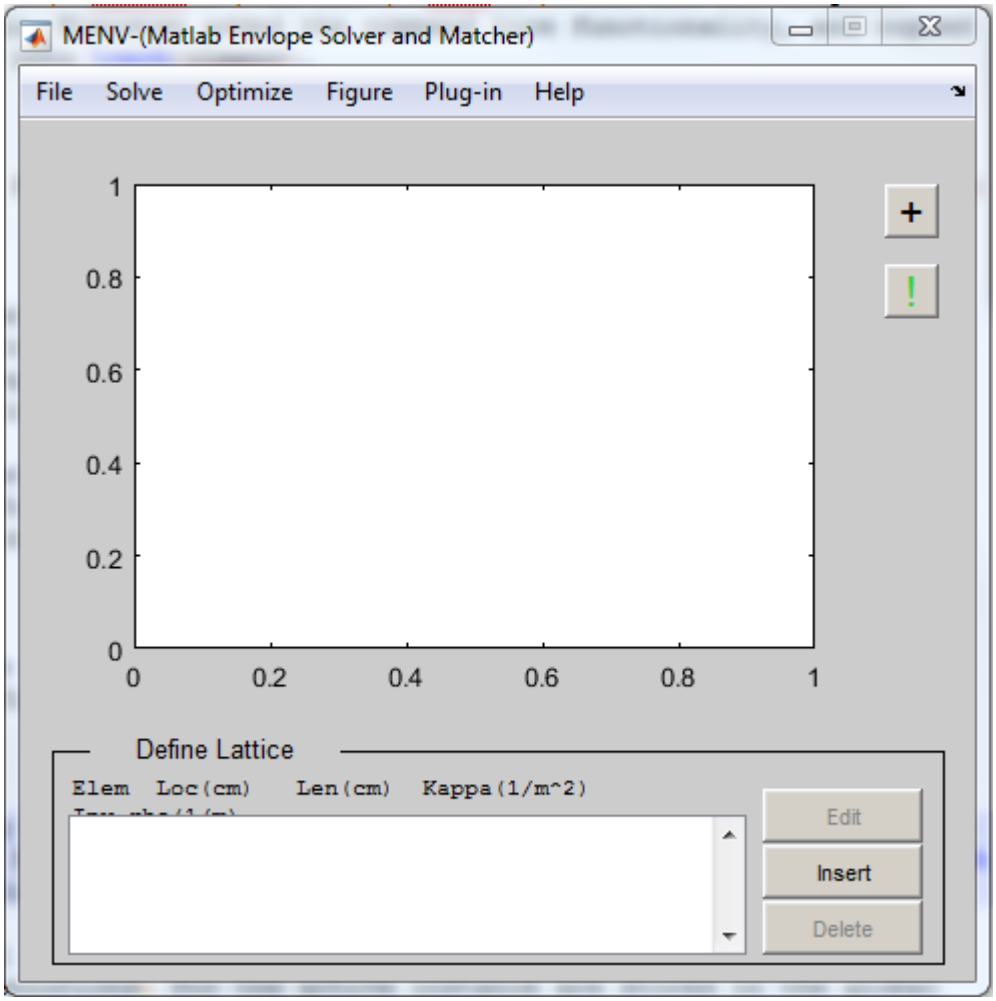

Figure 1: Main GUI window as it appears after launching MENV.

2 Running MENV from the GUI

The GUI can be launched by executing the command `menv` from the Matlab command line. Prior to running MENV, make sure that all files in the MENV directory are in the Matlab path (run `setpath` from the command line). This will launch a GUI window as shown in Fig. 1.

2.1 Defining and solving a lattice

Elements can be added using the “Insert" button, through a GUI interface shown in Fig. 2. Beam parameters (including initial conditions, emittance and perveance) are defined using the “Beam Parameters" option in the “Solve" drop-down menu. This launches the GUI shown in Fig. 3. Alternatively, run-files with .spt extension can be loaded using the command “Open" in the drop-down menu (or Ctrl-O).

Once beam and lattice parameters are defined, you can solve the envelope equations by hitting the solve button (green exclamation mark) or selecting “Compute Solution" from the “Solve" menu. To draw a schematic of the lattice elements, press the “draw" button. Figure 4 shows the solution for a 0.7 mA beam in a FODO lattice defined in the example file “FODO_0_7mA_dipl.spt."

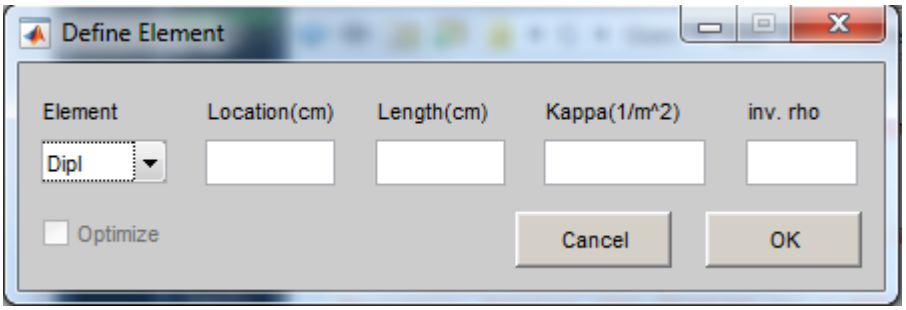

Figure 2: GUI window for inserting new elements.

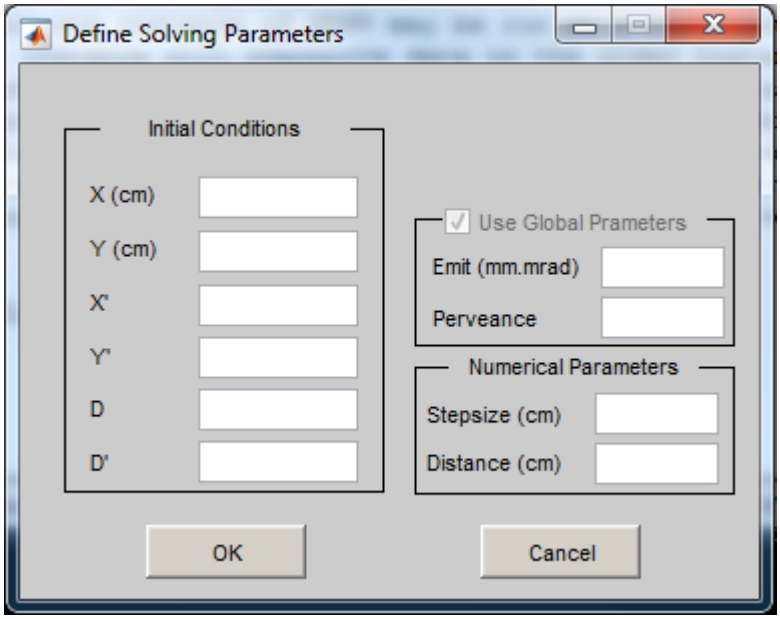

Figure 3: GUI window for defining beam parameters.

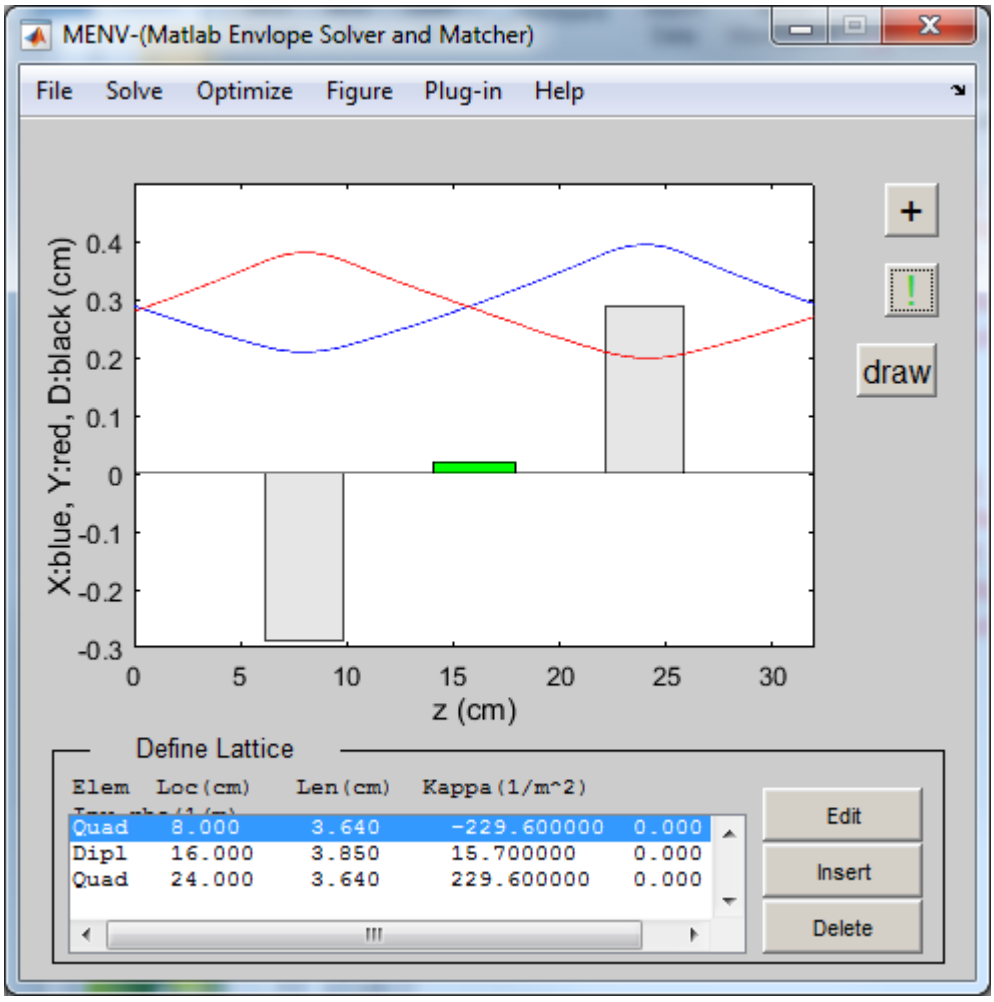

Figure 4: FODO lattice solution from example file “FODO_0_7mA_dipl.spt."

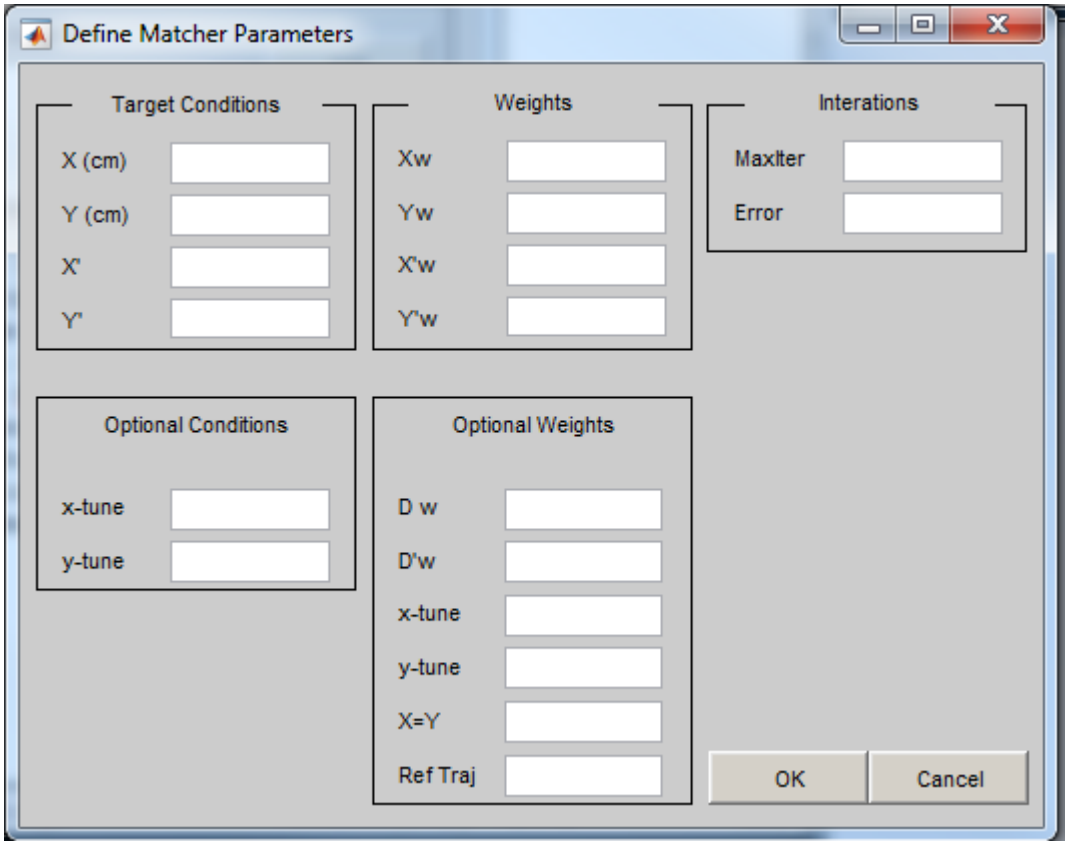

Figure 5: GUI window for defining parameters for matching/optimization routines.

2.2 Optimization in the GUI

MENV allows for two kinds of optimization: optimizing over initial beam conditions to find the periodic solution and optimizing quadrupole/solenoid strengths to meet a target condition. Before running a matching routine, you must define targets and weights by selecting “Matcher Parameters" under “Optimize." The GUI that is launched is shown in Fig. 5. If you are optimizing for the periodic solution, only the weights need to be defined. If you are optimizing to a target, targets must be defined as well. Targets and weights for X,Y,X’,Y’ at the end of the transport line must always be defined.

There are additional targets and weights that *may* be defined as desired; if not defined they will be set to a default value of 0. Optional targets include desired values of X and Y tunes. Optional weights include dispersion (use if you want to also find the periodic solution for dispersion), weights for the tune condition, and weights for global properties of the beam envelope. The global properties include a condition for “roundness" `X=Y`, which is defined as the rms value of `x(s)-y(s)` along the beam-line. The `Ref Traj` condition tries to match the envelope to a reference trajectory (inspired by the SPOT code, on which MENV is based [Allen, C. K., Guharay, S. K., & Reiser, M. (1998). Optimal Transport of low Energy Particle Beams. Epac, (3), 2324–2326.]). The reference trajectory is defined either by loading a text file in which a curve is defined as series of points in X-Z (“Plug-in," “Reference Trajectory," “Load") or by drawing in the MENV window (“Plug-in," “Reference Trajectory," “Draw").

2.3 Side note on parallel instances

The data (parameters, settings and solutions) for the active instance are stored in the global object `guim`. As this is a global variable that does not have a unique name for each instance of MENV, only one instance of MENV may be run at a time (you can launch two, but data from the most recent instance will overwrite data in the older instance). If you want to run parallel MENV instances, you need to run separate Matlab instances as well. This could be resolved by attaching data to the GUI window rather than a global variable, and this was the case in the older (Hui Li) version of MENV. The choice was made to adopt a global variable to allow use of functions written in the "command-line" MENV class `clmenv` from within the GUI (instead of having separate functions for command-line and GUI versions).

3 Organization of MENV

3.1 Main MENV routines

The two MENV interfaces (GUI or command-line) draw on the same set of functions to perform envelope integration and optimization. Table 1 describes the functions which make up the heart of the MENV code. All of these are found in the folder "menv-main/."

Table 1: MENV functions

Name	Purpose
clmenv	Class used to run command-line and GUI menv. Contains many useful routines to load, organize and store simulation data and parameters.
runmenv	Function which runs MENV integrator and returns solution data as a function of s. Sub-routine for "Solve" method in clmenv and GUI.
menv_makekappaarray	Translates lattice definition into κ-array used in integrator Returns strength vector for optimizable elements Sub-routine for `runmenv` function
menv_updatekappaarray	Updates κ-array based on changes to optimizable elements Returns new strength vector for optimizable elements Sub-routine for `optfunc` function
menv_integrator	Lower-level function to perform integration through defined beamline
match2period	Function to optimize initial beam conditions for periodic matched solution Sub-routine for "Periodic Matcher" method in clmenv and GUI.
match2target	Function to optimize focusing elements to meet specified target Sub-routine for "Target Matcher" method in clmenv and GUI.
optfunc	Function used in each optimization step in "match2target" Input: strength of optimizable focusing elements Output: weighted cost-function vector
periodfunc	Function used in each optimization step in "match2period" Input: initial conditions X,X',Y,Y',D,D' Output: weighted cost-function vector

3.2 GUI code

The dependencies of the GUI `menv` are listed below. All GUI-specific files are stored in the folder "menv-gui/".

- menv.m :: Code for creating main GUI window
- defElement.m :: Code for creating element insert/edit GUI window
- defParam.m :: Code for creating beam parameter GUI window
- defMatcher.m :: Code for creating matcher parameter GUI window
- ImportOpt.m ::
- prepareMenu.m :: Code for creating menus on main GUI window
- menvEvent.m :: Contains code for interpreting commands sent by the GUI window
- clmenv.m :: Contains many of the upper level routines, called by some of the commands in menvEvent.m

It should be noted that GUI MENV was written following (presumably) older Matlab guidelines in which GUI data is stored in a .mat file as opposed to a .fig file.

3.3 Additional (untested) routines

I wrote additional optimization routines to try to solve more sophisticated problems. These have not been vetted as thoroughly and should be closely scrutinized before use. To indicate this, they are stored in the folder "beta/" in the MENV directory. There may be some bugs due to changes to the main MENV routines/structure that were not filtered through to these functions. Matlab has a huge library of optimization algorithms (depending on the license), these can be added modularly to expand MENV capabilities in a similar fashion as seen here.

Table 2: "beta" MENV functions

Name	Purpose
GSmatch2target	Function to optimize focusing elements to meet specified target using Matlab "Global Search" algorithm. Sub-routine for "Global Search" method in clmenv and GUI.
MOmatching	Function to optimize focusing elements to meet specified target using a Multi-Objective algorithm. Sub-routine for "Multi-Objective" method in clmenv and GUI.
svoptfunc	Function used in each optimization step in "GSmatch2target" Calls "optfunc" but returns cost function as single-vector rms quantity
stepfuncMO	Function used in each optimization step in "MOmatcher"

Also in the "beta/" folder, I include routines "skewstep.m" and "skew_test.m." Apparently at some point there was an attempt to include skew-coupling terms in the envelope integration. I cannot speak to the accuracy or robustness of this code, but I preserve it in case it is ever pursued again.

3.4 UMER utilities

Some routines specific to UMER have been added for convenience. These are stored in folder "UMER-utils/". The Current2Kappa and converse function allow for the specification of elements 'Y' (enlarged Y-section quad) and 'P' (enlarged Y-section dipole) and will use appropriate conversion factors for the quadrupole focusing gradients in each. This is beyond the standard elements 'D', 'Q', and 'S' (dipole, quad, solenoid). For all elements, it is up to the user to specify the correct hard-edged model lengths.

Table 3: UMER utilities included in MENV

Name	Purpose
Current2Kappa	Convert Current [A] to Kappa value [m^{-2}]. Uses Bernal 2006 numbers for HE magnets.
Kappa2Current	Converse of above
Gauss2Kappa	Convert field value [Gauss] to Kappa value [m^{-2}].
Kappa2Gauss	Converse of above
getBeamMenvParam	Returns initial conditions for UMER beams based on python script Ubeams.py used in WARP decks. contributed by Ben Cannon

4 MENV from the command line

A simple example, in which a .spt file is loaded into MENV and run from the command line (modified from run_clemnv.m):

```
% -- initiate clmenv instance
clm = clmenv();

% -- load .spt file
sptfilename = which('FODO_0_7mA_dipl.spt');
clm.open(sptfilename)

% -- solve
clm.solve()

% -- find periodic solution
clm.periodicmatcher()
```

A slightly more complex example, in which a lattice is constructed in MENV from the command line (modified from run_clemnv_inj.m):

```
clm = clmenv();

% -- set params

% Beam parameters
emittance = 30; % mm-mrad
perveance = 0.0001045;
x0        = .3; % cm
y0        = .3; % cm
xp0       = 0.0;
yp0       = 0.0;

% Simulation parameters
stepsize = 0.005; % cm
distance = 32; % cm

% optimization parameters
iterations = 20;
tolerance  = 1e-6;

%-- Lattice description
elements ='QDQ'; % element types
location = [8,16,24]; % element location
lengths = [3.6400,3.8500,3.6400]; % element length
str = [220,15.7000,-220]; % quadrupole strength (kappa)
bend_ang = [0 10 0]*pi/180; % bend angle
invrho = bend_ang./lengths; % inverse rho
opt = [0,0,0];

% -- assign params to menv structures
clm.maketarget([1,1,1,1],[1,1,1,1])
clm.makeoptiset(iterations,tolerance)  % Makes file optiset
clm.defmatcher() % loads matcher settings

ic = struct();
ic.x0 = x0; ic.y0=y0; ic.xp0=xp0; ic.yp0=yp0; ic.D0 =0 ; ic.Dp0 = 0;
clm.makeparams(emittance,perveance,ic,stepsize,distance) % makes param file
clm.defparam() % loads params
```

```
39
40 clm.deflattice(elements,location,lengths,str,opt,invrho) % loads lattice
41
42 % -- look at initial solution
43 clm.solve()
44
45 % -- solve for periodic solution
46 clm.periodicmatcher()
```