Bow Simulator v0.4
User Manual

Contents

1 Introduction **2**

2 The Bow Editor **3**
- 2.1 Profile 4
- 2.2 Width and Height 5
- 2.3 Material 6
- 2.4 String 7
- 2.5 Masses 8
- 2.6 Operation 8
- 2.7 Comments 9
- 2.8 Settings 9

3 Simulation Results **11**

4 Command Line Interface **13**

5 Background Information **14**
- 5.1 A Simple Bending Test 14
- 5.2 The Internal Bow Model 15
- 5.3 Validation of Simulation Results 16
  - 5.3.1 Statics of a straight steel bow 16

A Input File Structure **19**

B Output File Structure **20**

C Python Scripting Example **21**

1 Introduction

About this Manual

This is the User Manual for Bow Simulator, a software tool for bow and arrow physics simulation. It shows how to use the program, explains all the input and output data and also contains some related background information.

For the latest version of the software and this manual visit
http://bow-simulator.org/.

Support

If you need help, want to report a problem or give feedback you can either use the mailing lists on the website or contact the author directly at
s-pfeifer@gmx.net.

2 The Bow Editor

The bow editor is the main window of the application. Here you can load, modify and save bow models and start static and dynamic simulations.

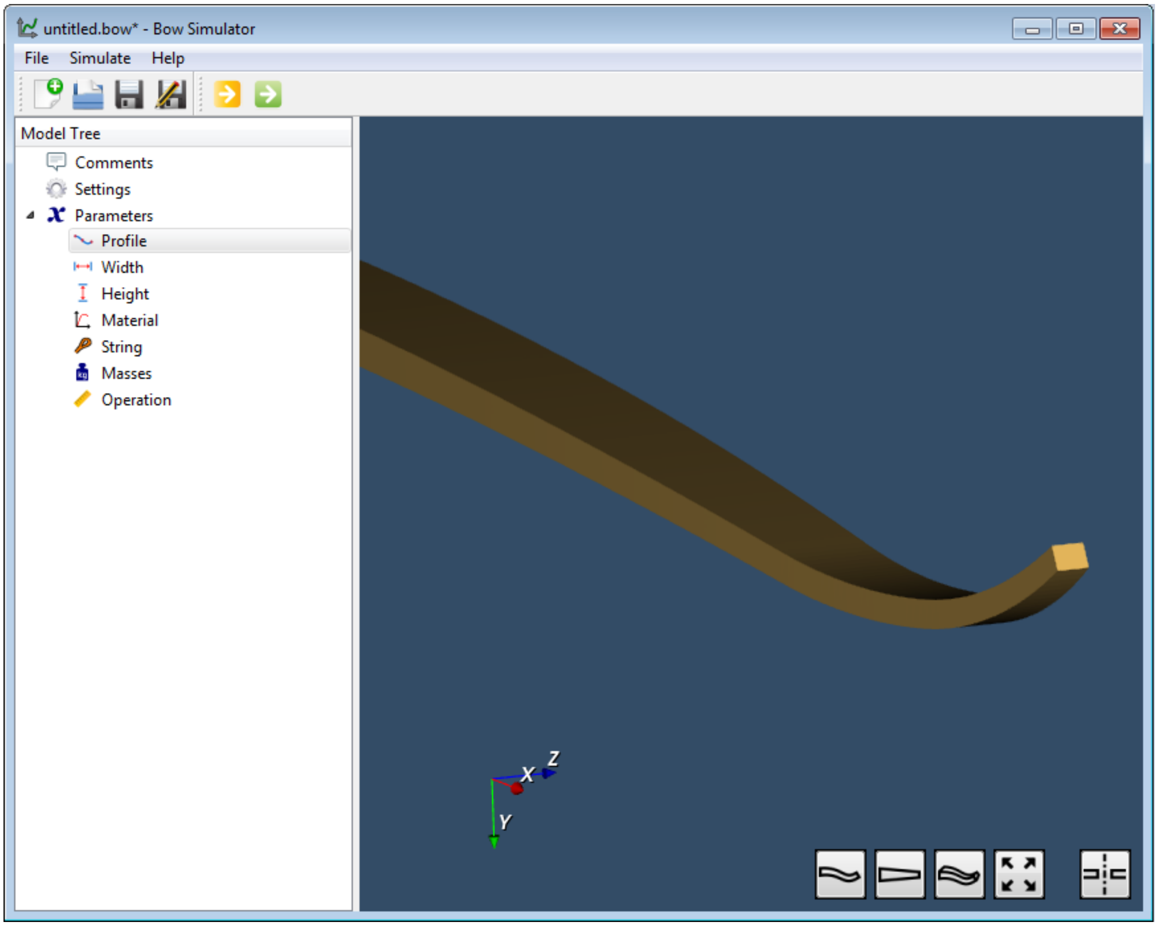


Figure 1: Bow Editor

Use the model tree on the left to change the properties of the bow. Double-clicking on any of the items opens an associated input dialog. Those dialogs will be explained in more detail in the following sections.

The 3d view on the right side shows the current limb geometry. Use the mouse to rotate, zoom and shift the view. More view options are available through the buttons on the bottom-right corner.

2.1 Profile

The profile curve determines the shape of the back of the bow in unbraced state. Use the table on the left to edit the profile and check the result on the plot on the right.

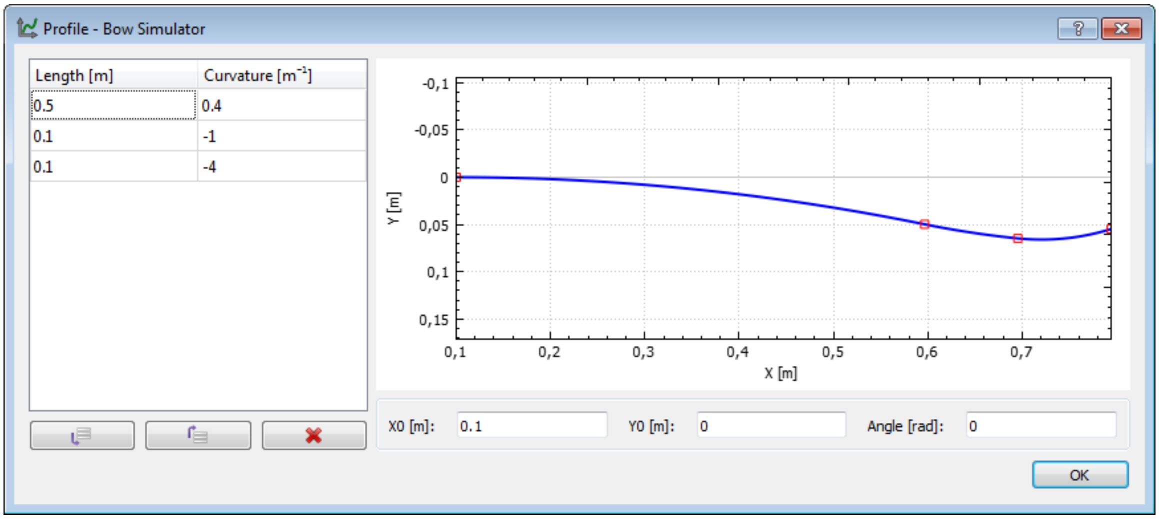


Figure 2: Profile dialog

The profile curve consists of a series of connected arc segments that each have a certain length and curvature. Each row in the table defines one such arc segment. You can add and remove segments by using the buttons below the table.

Note: Mathematically, the curvature κ of a segment is the reciprocal of it's radius r, so you can calculate the curvature via $\kappa = \frac{1}{r}$ if you know the radius. A curvature of zero corresponds to a straight line.

On the bottom right you can specify x-, y- and angular offsets that control the starting point and orientation of the limb. This can be used to account for a stiff middle section/riser.

2.2 Width and Height

With the width and height dialogs you can define the limb's cross sections along its profile curve.

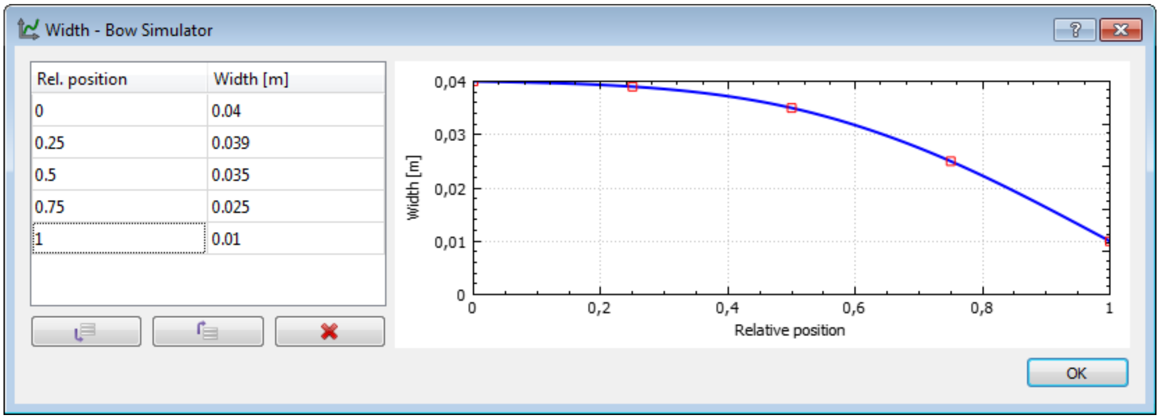


Figure 3: Width dialog

Use the table on the left to specify width or height values at certain relative positions along the limb. A smooth curve (cubic spline) passing through the supplied values is constructed, which you can see on the plot on the right.

The relative positions define the location along the profile curve. They can be anything you want, for example values between 0 and 1 or percentages. They are mapped to the profile curve's arc length in a linear way, such that the first and last position corresponds to the beginning and end of the curve respectively.

Example: If you have a profile curve with a total length of 0.8m and you define three relative positions as 7, 8 and 9 then the cross section properties are placed at arc lengths 0m (beginning), 0.4m (middle), and 0.8m (end).

This definition of the cross section positions relative to the total arc length of the profile curve makes it possible to change the profile curve without having to redefine all cross sections.

2.3 Material

Here you can specify the properties of the limb material.

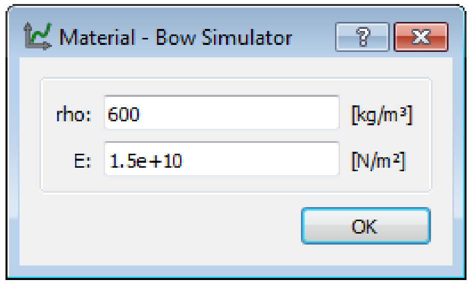


Figure 4: Material dialog

Two material constants are needed:

- **rho:** Density (Mass per unit volume)
- **E:** Elastic modulus (Measure for the stiffness of a solid material)

For manufactured materials like e.g. steel or fiberglass you might can often find those numbers in a datasheet provided by the manufacturer.

Wood is a bit more difficult, because the properties can vary quite a bit even within the same species of tree. You can find average numbers on the internet, for example at `http://www.wood-database.com`. Most of the time those are probably good enough as a first reference. However, in order to be really sure about a specific piece of wood there is no other way than to test it. One possibility is a bending test as shown in section 5.1.

2.4 String

Here you can define the mechanical properties of the string by providing data for the string material and the number of strands being used.

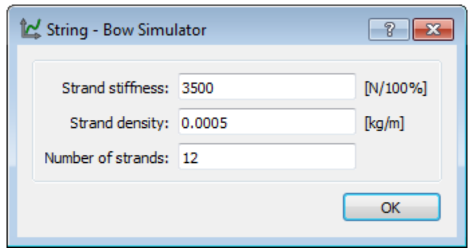


Figure 5: String dialog

- **Strand density:** Linear density of the strands (mass per unit length)
- **Strand stiffness:** Stiffness of the strands against elongation (force per unit strain)
- **Number of strands:** Total number of strands in the string

The linear density of a string material can be easily determined with a kitchen scale (weight divided by length), but the stiffness is much more difficult to obtain. Table 1 shows some reference values taken from the SuperTiller V6.6 Excel spreadsheet by Alan Case.

Material	**Stiffness** [N/100%]	**Density** [kg/m]	**Breaking strength** [N]	**Source/Comment**
Dacron B50	3113.76	0.000333	217.96	BCY assuming 7% elongation at break (linearized)
Fast Flight	14086.04	0.000182	422.58	BCY assuming 3% elongation at break (linearized)
Dyneema	18860.46	0.000160	578.27	Calculated assuming linear stress-strain (to break)
Linen 40/3	1668.08	0.0642	49.38	Maurice Taylor, Archery The Technical Side, 1947
Silk	1026.51	0.0930	65.83	Maurice Taylor, Archery The Technical Side, 1947

Table 1: Properties of common bowstring materials according to SuperTiller V6.6

Note: The stiffness of the string can be important in dynamic analysis. The static results however aren't affected very much by it as long as it is high enough to prevent significant elongation.

2.5 Masses

These are used to account for the various dead weights that can be attached to a bow.

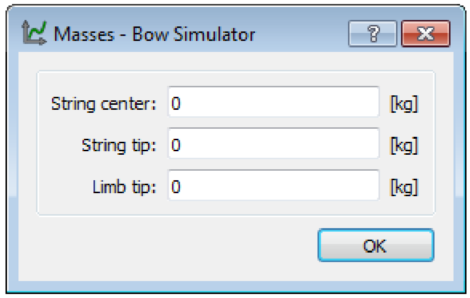


Figure 6: Masses dialog

- **String center:** Additional mass at the string center (serving, nocking point)
- **String tip:** Additional mass at the ends of the string (serving, silencers)
- **Limb tip:** Additional mass at the limb tips (overlays and the like)

2.6 Operation

Here you can find all parameters that define the conditions under which the bow it operates.

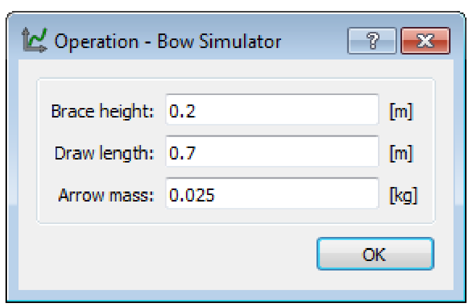


Figure 7: Operation dialog

- **Brace height:** Distance of the string center to the coordinate origin ($x = 0$, $y = 0$) in braced state
- **Draw length:** Distance of the string center to the coordinate origin in fully drawn state
- **Arrow mass:** Total mass of the arrow

2.7 Comments

The comments are meant for documenting the bow model. Any notes about the bow and the simulation results can be added here.

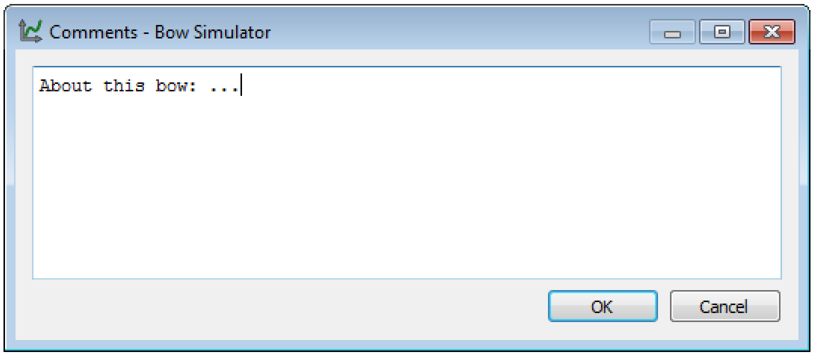


Figure 8: Comments dialog

2.8 Settings

These are numerical settings used by the simulation. You can do some fine-tuning here, but most of the time the default values should be fine. However, as the default values favor accuracy and realiability over performance there might be use cases where finding faster settings is worth it. Think about running a large number of simulations in batch mode for example.

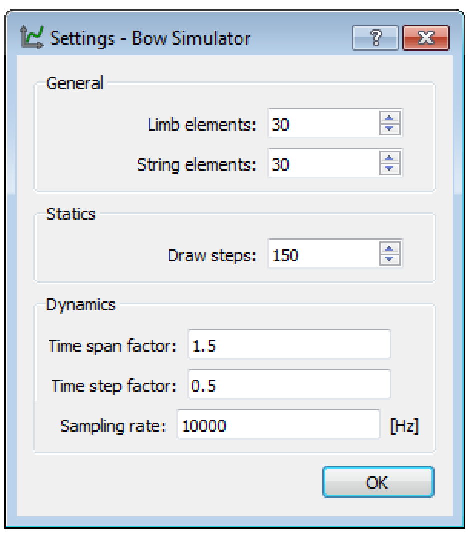


Figure 9: Settings dialog

These are the individual settings:

- **General**
  - **Limb elements:** Number of finite elements that are used to approximate the limb. More elements increase the accuracy but also the computing time.
  - **String elements:** Same as above.
- **Statics**
  - **Draw steps:** Number of steps that are performed by the static simulation from brace height to full draw. This determines the resolution of the static results. You can usually decrease this value to speed up the simulation.
- **Dynamics**
  - **Time span factor:** This controls the time period that is simulated. A value of 1 corresponds to the time at which the arrow reaches brace height. The default value is larger than that in order to capture some of the things that occur after the arrow left the bow (for example the maximum dynamic loads on limb and string).
  - **Time step factor:** When carrying out the dynamic simulation the program will repeatedly use the current state of the bow at time t to calculate the next state at time $t + \Delta t$ where Δt is some small timestep. This timestep has to be chosen small enough to get an accurate and stable solution but also as large as possible to keep the total number of steps low. The program can estimate this optimal timestep, but to be on the safe side the estimation is reduced by a safety factor between 0 and 1 that you can choose here.
  - **Sampling rate:** Limits the time resolution of the output data. This is done because the dynamic simulation usually produces much finer grained data than is actually useful. Not including all of that in the final output saves memory and computing time.

3 Simulation Results

You can start static or dynamic simulations by using the yellow and green toolbar buttons or the simulation menu. Once the simulation has finished, a new window with the results will open.

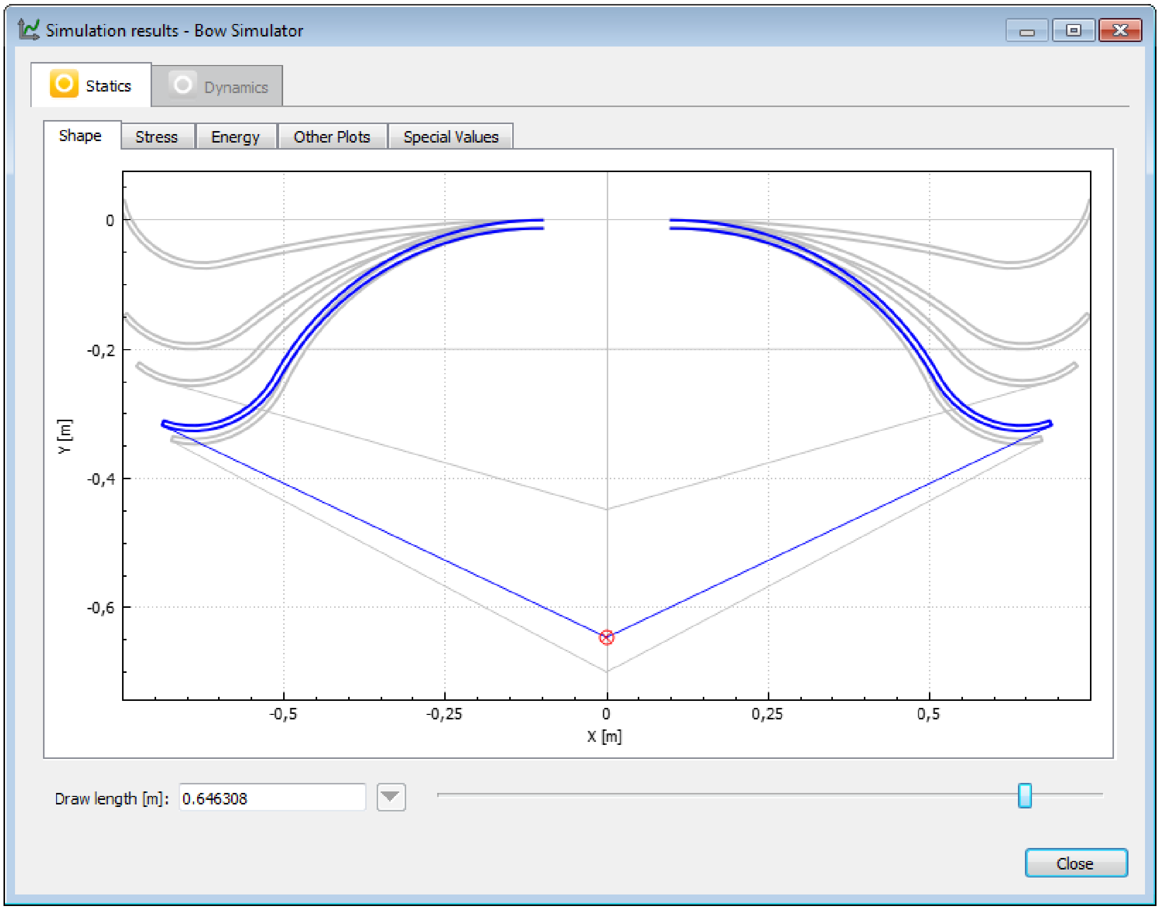


Figure 10: Simulation results

At the top of the result window you can switch between the static and dynamic (if available) results. The results themselves are organized in different tabs. At the bottom of the window there is a slider where you can change the current draw length (statics) or time (dynamics). This value applies to all of the result tabs.

These are the individual result tabs:

- **Shape:** Shows the shape of the limb and string as well as the position of the arrow at different stages of either the draw (statics) or the shot (dynamics)
- **Stress:** Shows the distribution of material stress (back and belly) along the length of the limb
- **Energy:** Shows how potential and kinetic energy of the different parts of the bow develop during the simulation
- **Other Plots:** This tab is for anything not contained in the default results. Here you can just combine simulation results arbitrarily and plot them together.
- **Special values:** Most of these numbers characterize some aspect of the bow's performance. They are different for the static and dynamic case.
  - **Statics:**
    - **String length:** Length of the string (calculated such that the bow meets the brace height specified by the user)
    - **Final draw force:** Draw force in fully drawn state
    - **Drawing work:** Total work done by drawing the bow. This is equal to the area under the draw curve.
    - **Storage ratio:** This is an indicator of the bow's capability to store energy and is defined (/made up by the author) as

      $$\text{storage\_ratio} = \frac{\text{drawing\_work}}{1/2 \cdot \text{draw\_force} \cdot (\text{draw\_length} - \text{brace\_height})}.$$

      It describes the amount of energy stored by the bow's draw curve in relation to a fictious linear draw curve with the same final draw force.
  - **Dynamics:**
    - **Arrow velocity:** Velocity of the arrow when leaving the bow
    - **Arrow energy:** Kinetic energy of the arrow when leaving the bow
    - **Efficiency:** Degree of efficiency of the bow. Useful energy output (kinetic energy of the arrow) divided by energy input (static drawing work).

4 Command Line Interface

The command line interface can be used to start simulations in batch mode, without opening the GUI. This way Bow Simulator can be called from other programs for performing more advanced computations like parameter studies and optimizations.

The command line parameters are as follows:

bow-simulator [input] [output] [options]

- **input:** Path to an input file (`.bow`)
- **output:** Path for the output file (`.dat`)
- **options:** Simulation options, `--static` or `--dynamic`

All of the arguments are optional. Calling Bow Simulator with either no arguments or only an input file will open the GUI. Otherwise if either an output file or simulation options are provided, the simulation is carried out silently. If no output file was specified a default one with the same name as the input file is created.

Note: To use the command line interface on Windows you have to either use the complete path to the `bow-simulator.exe` executable or add the installation directory to your `PATH` environment variable.

Input and Output Formats

Bow Simulator's input files use the JSON[1] format to represent their data. JSON is a human readable text format that stores different types of data in a hierarchical way. The output files use MessagePack[2], a more compact binary format that is otherwise very similar to JSON.

Both JSON and MessagePack are very common formats with implementations available in many programming languages. An example for using Bow Simulator with Python is shown in Appendix C.

The exact structure and definition of the data fields contained in the input and output files is documented in Appendix A and B, respectively.

Note: The structure of the input and output files is not yet stable. Future versions of Bow Simulator will very likely introduce some breaking changes.

[1] `http://json.org/`
[2] `http://msgpack.org/`

5 Background Information

5.1 A Simple Bending Test

A bending test is an easy way to determine the elastic modulus of a material. It can be done without any special equipment. Figure 11 shows the setup. A test piece with length l is clamped on one side and subjected to a vertical force F at its free end. The deflection s due to this load is measured.

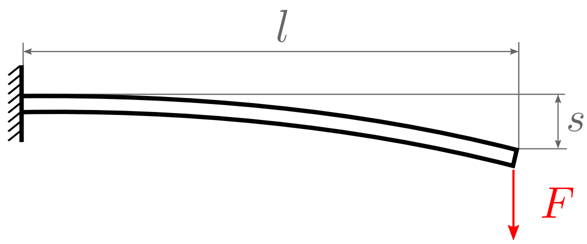


Figure 11: Experimental setup

The elastic modulus can then be calculated depending on the cross section geometry using the equations in Table 2.

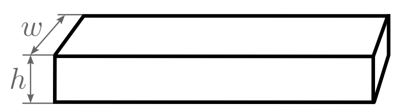
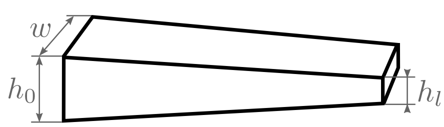
Test geometry	Elastic Modulus
	$E = \dfrac{4}{wh^3}\,\dfrac{Fl^3}{s}$
	$E = \dfrac{12\,\ln(h_l\,l) + 6}{w\,(h_l - h_0)^3}\,\dfrac{Fl^3}{s}$

Table 2: Elastic modulus for different test geometries

Here are a few practical considerations:

- The precision of the cross sections is very important, especially the height.
- The equations above hold for slender beams and small deflections. The test setup should be chosen accordingly. As a rule of thumb: $h, s < l/15$.
- A simple way to apply a defined force is to hang a mass m onto the beam tip and use $F = m \cdot g$, with $g = 9.81\,\mathrm{m/s^2}$.
- If there is some small initial deflection due to gravity, then s is simply the difference in deflection after application of the force.

5.2 The Internal Bow Model

This section is intended to give interested users an overview of the mathematical bow model behind Bow Simulator, i.e. how the different components of the bow are modeled and what the assumptions and limitations are. This section will eventually be replaced by a separate technical documentation of the simulation model.

Limb: The limb is regarded as an Euler-Bernoulli beam. This means that all cross-sections of the beam are assumed to stay flat and perpendicular to the beam axis during deformation. The Euler-Bernoulli beam theory therefore only accounts for bending deformation and neglects shear deformation, which is usually a valid thing to do for long, slender beams.

The material of the limb is considered linear-elastic, so the relation between material stress σ and strain ϵ at any point in the limb is given by the linear equation $\sigma = E \cdot \epsilon$ with the elastic modulus E as a material constant. The overall behaviour of the limb however is nonlinear due to the nonlinear kinematics/geometry of large deformations.

String: Contrary to the limb, the string only transfers longitudinal forces and has no flexural rigidity. The material is considered linear-elastic as well. The string has a constant cross section and is internally implemented as a chain of point masses connected by springs. Additional point masses at the center and the tips represent things like servings and nocking point.

Arrow: The arrow is modeled as a point mass. Deformation and vibration of the arrow (known as *archers paradox*) is neglected/not captured by this model. That's because the scope of this program is only to evaluate overall bow performance, things like final arrow velocity, degree of efficiency, etc. For this purpose a point mass is sufficient.

Symmetry: The bow is assumed to be symmetric. This is often only an approximation as most bows besides crossbow prods are actually slightly asymmetric. The assumption of symmetry simplifies the definition of the parameters by the user (no need to define the limb twice). It also allows the program to simulate only one half of the bow, which reduces the computing time. (As a user you don't have to take this into account, all input and output data of the program corresponds to the complete bow.)

5.3 Validation of Simulation Results

A very important task in the development of this software is to make sure that the results obtained by simulation agree reasonably well with real world examples. This section shows the validation efforts made so far. It is still very sparse, so if you have used this program for a real world application, let me know about your results and they will be added here.

5.3.1 Statics of a straight steel bow

In this experiment the draw curve and limb shapes of a small steel bow with a constant cross section have been measured and compared to version 2014.4 of Bow Simulation Tool (now Bow Simulator). The bow is shown in figure 12 and has been made from an old saw blade. Steel is a good material for this kind of test because of its homogenous mechanical properties. The elastic modulus was assumed to be $E = 210\ \mathrm{GPa}$ which is a good estimate for most types of steel.

Figure 12: Steel bow. Cross section: 16.85 $\times$ 0.75mm. Length: 269mm. Brace heighth: 49.8mm

The experiment was carried out by hanging a plastic bag at the string center. The draw force was then gradually applied by counting steel balls with a known mass into the bag. After every load step the draw length was measured and a photo of the bow was taken.

Figure 13 shows a comparison between the measured and the simulated draw curve and figure 14 compares the pictures of the limb against the simulated limb shapes.

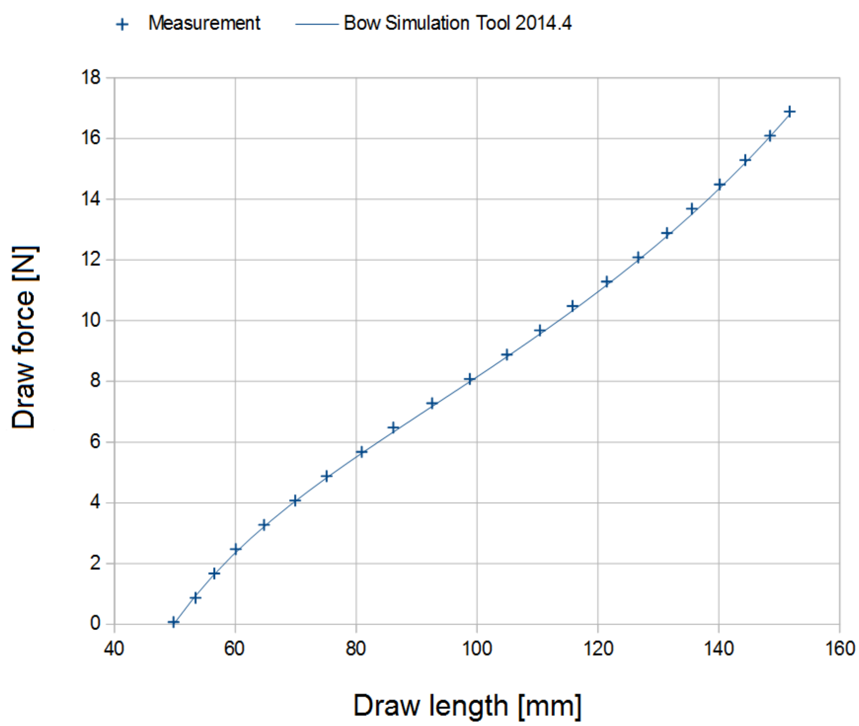


Figure 13: Experimental and simulated draw curves

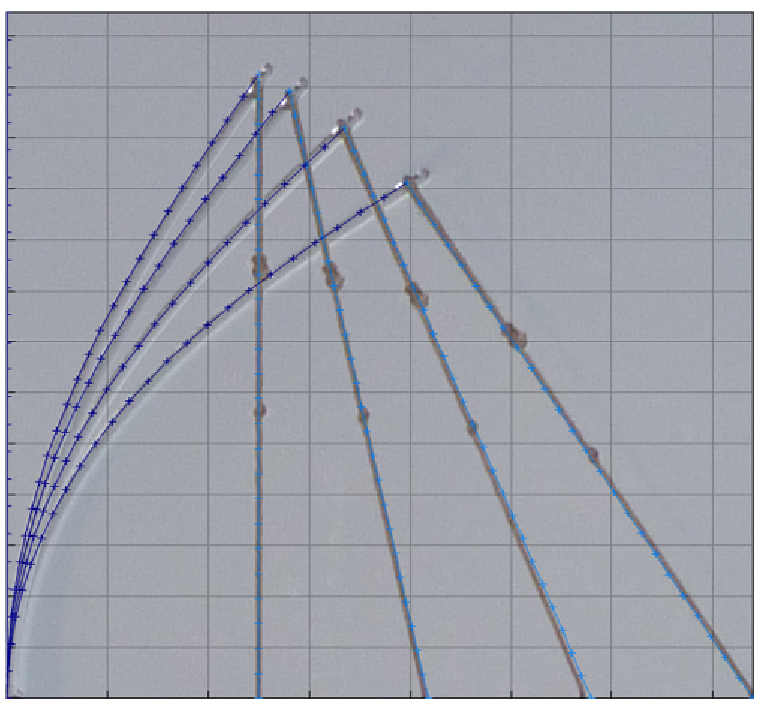


Figure 14: Experimental and simulated limb shapes

The agreement between experiment and simulation is surprisingly good here. It really shows the potential of such kinds of simulations, provided that the material properties are well known and a the bow can be built exactly as simulated, with low tolerances.

But this is still a very simple example. The next step would be to repeat this experiment with bows that have varying cross sections and non-straight profiles. Another open question are the dynamic simulation results. It's unclear if they can match experiments as good as the static results do, because there are much more uncertainties involved.

A Input File Structure

Field	Type	Unit	Description
meta			
version	`string`	–	Internally used version string
comments	`string`	–	User comments
settings			
n_elements_limb	`integer`	–	Number of limb elements
n_elements_string	`integer`	–	Number of string elements
n_draw_steps	`integer`	–	Number of steps for the static simulation
time_span_factor	`double`	–	Factor for modifying total simulation time
time_step_factor	`double`	–	Factor for modifying simulation time steps
sampling_rate	`double`	Hz	Time resolution for the dynamic output
profile			
segments			
args	`double[]`	m	Array of segment lengths
vals	`double[]`	m^{-1}	Array of segment curvatures
x0	`double`	m	X offset of the profile curve
y0	`double`	m	Y offset of the profile curve
phi0	`double`	rad	Angular offset of the profile curve
width			
args	`double[]`	–	Array of relative positions
vals	`double[]`	m	Array of cross section widths
height			
args	`double[]`	–	Array of relative positions
vals	`double[]`	m	Array of cross section heights
material			
rho	`double`	kg/m^3	Mass density of the limb material
E	`double`	Pa	Elastic modulus of the limb material
string			
strand_stiffness	`double`	N	Stiffness of the string material
strand_density	`double`	kg/m	Linear density of the string material
n_strands	`double`	–	Total number of strands
masses			
string_center	`double`	kg	Additional mass at string center
string_tip	`double`	kg	Additional mass at string tips
limb_tip	`double`	kg	Additional mass at limb tips
operation			
brace_height	`double`	m	Brace height
draw_length	`double`	m	Draw length
mass_arrow	`double`	kg	Arrow mass

B Output File Structure

`N`: Number of simulation steps
`P`: Number of limb nodes
`Q`: Number of string nodes

Field	Type	Unit	Description
setup			
limb			
s	`double[P]`	m	Arc lengths of the limb nodes (unbraced)
x	`double[P]`	m	X coordinates of the limb nodes (unbraced)
y	`double[P]`	m	Y coordinates of the limb nodes (unbraced)
string_length	`double`	m	Length of the unstressed string
statics			
states			
[...]			Sequence of bow states. See Table below.
final_draw_force	`double`	N	Final draw force
drawing_work	`double`	J	Drawing work
storage_ratio	`double`	–	Storage ratio
dynamics			
states			
[...]			Sequence of bow states. See Table below.
final_arrow_velocity	`double`	m/s	Final velocity of the arrow
final_arrow_energy	`double`	J	Final energy of the arrow
efficiency	`double`	–	Degree of efficiency

Field	Type	Unit	Description
states			
time	`double[N]`	s	Time
draw_length	`double[N]`	m	Draw length
draw_force	`double[N]`	N	Draw force
string_force	`double[N]`	N	String force (total)
strand_force	`double[N]`	N	String force (strand)
grip_force	`double[N]`	N	Grip force
x_limb	`double[N][P]`	m	X coordinates of the limb nodes
y_limb	`double[N][P]`	m	Y coordinates of the limb nodes
phi_limb	`double[N][P]`	rad	Rotation angles of the limb nodes
x_string	`double[N][Q]`	m	X coordinates of the string nodes
y_string	`double[N][Q]`	m	Y coordinates of the string nodes
sigma_back	`double[N][P]`	Pa	Limb stress at the back
sigma_belly	`double[N][P]`	Pa	Limb stress at the belly
pos_arrow	`double[N]`	m	Arrow position
vel_arrow	`double[N]`	m/s	Arrow velocity
acc_arrow	`double[N]`	m/s^2	Arrow acceleration
e_pot_limbs	`double[N]`	J	Potential energy of the limbs
e_kin_limbs	`double[N]`	J	Kinetic energy of the limbs
e_pot_string	`double[N]`	J	Potential energy of the string
e_kin_string	`double[N]`	J	Kinetic energy of the string
e_kin_arrow	`double[N]`	J	Kinetic energy of the arrow

C Python Scripting Example

The code below shows how Bow Simulator can be used with the Python programming language. It loads, modifies and saves an input file, runs a simulation with it and prints one of the results.

Python can load and save `.bow` files out of the box with the `json` standard library module. The output files are loaded with the external `msgpack-python`[3] package. It can be installed via

```
pip install msgpack-python
```

Bow Simulator itself can be called like any other command line program either with `os.system` or `subprocess.call`.

```python
import json
import msgpack
import os

# Load input file
with open("input.bow", "r") as file:
    input = json.load(file)

# Modify input
input["string"]["n_strands"] += 1

# Save modified input
with open("input.bow", "w") as file:
    json.dump(input, file)

# Run a static simulation
os.system("bow-simulator input.bow output.dat --static")

# Load the output file
with open("output.dat", "rb") as file:
    output = msgpack.unpackb(file.read())

# Print a specific result
print(output["statics"]["final_draw_force"])
```

[3] `https://pypi.python.org/pypi/msgpack-python/`