Bow Simulator v0.3
User Manual

Contents

1 Introduction

About this Manual

This is the User Manual for Bow Simulator, a software tool for bow and arrow 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.sourceforge.net.

Support

If you need help, want to give feedback or report a problem 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 first thing you see when starting Bow Simulator. Here you can load, modify and save bow models and run static and dynamic simulations.
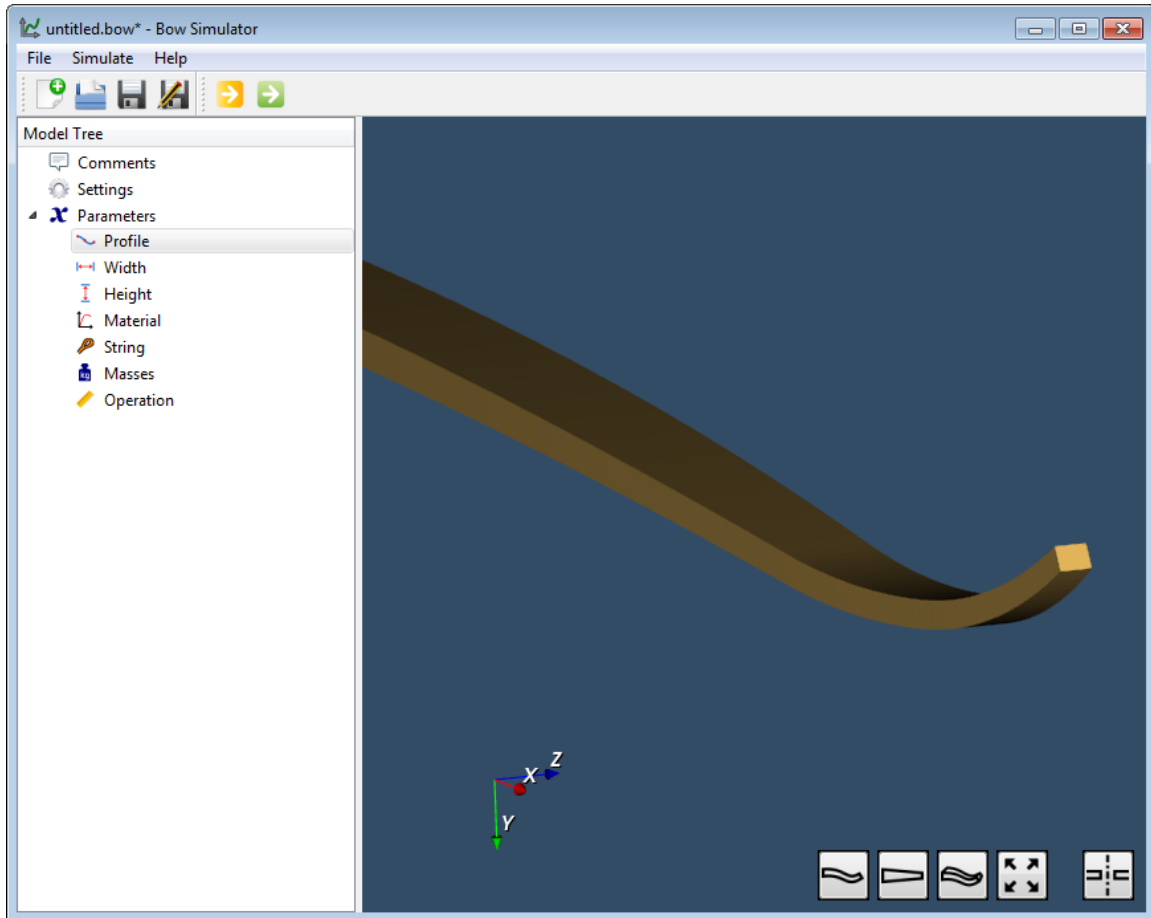
Figure 1: Bow Editor

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

The 3d view on the right-hand 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.

4

2.1 Comments

The first item in the model tree are the comments. Those are meant purely for documentation. Any notes or remarks about the bow and the simulation results can be added here.
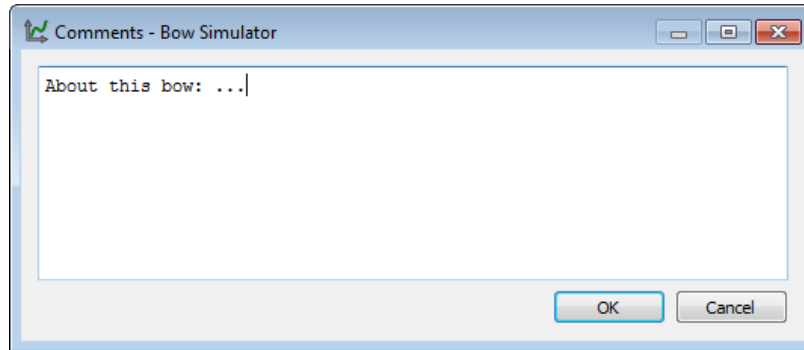

Figure 2: Comments dialog

2.2 Settings

These are the numerical settings used by the simulation. You can do some fine-tuning here and change the balance between accuracy and computing time. For most use cases the default values should be fine though.
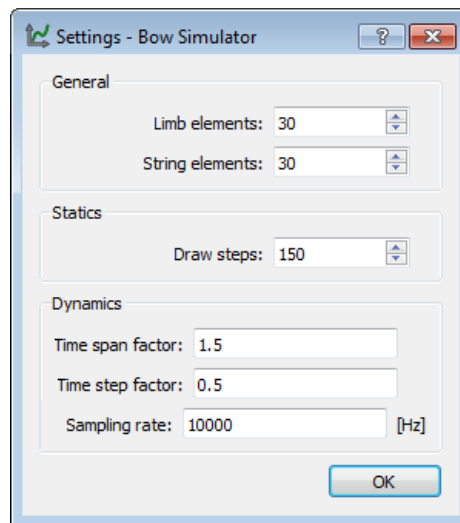

Figure 3: Settings dialog

Here is what the different parameters mean:

- **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. If you're only interested in the fully drawn state or the dynamics you could set this to something very low.

- **Dynamics**

  - **Time span factor:** This value 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:** To understand this parameter, some details about the dynamic solution method are necessary. When carrying out the simulation the program will 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 is repeated over and over until the desired time span has been simulated. The timestep Δt has to be chosen small enough to get an accurate and stable solution (that doesn't "explode" numerically) but also as large as possible to keep the number of steps that have to be performed low. The program can calculate a crude estimation for the optimal timestep, but to be on the safe side this estimation is reduced by a safety factor between 0 and 1 that you can manipulate here. The default value is relatively low, favouring robustness over perfromance. Usually the simulation can be sped up by increasing this value. On the other hand, if the dynamic results are garbage you could try decreasing it.

  - **Sampling rate:** Upper limit for 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 both memory and computing time.

Note: Many of those numerical settings control a tradeoff between accuracy on one hand and computing time on the other. A general rule of thumb for finding a good value for any of those is to keep increasing accuracy until the simulation results you're interested in do not change significantly anymore.

2.3 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.
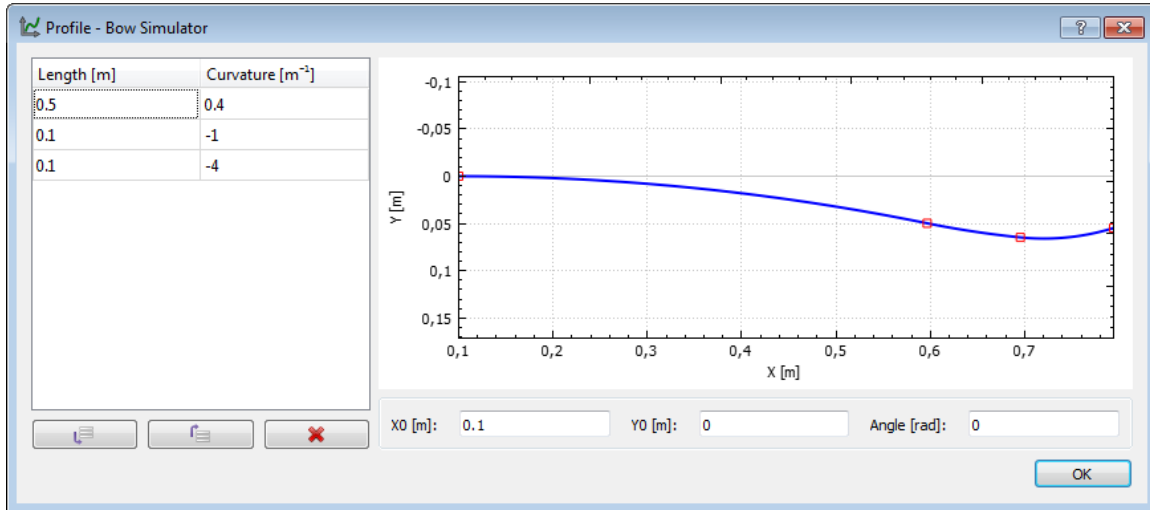
Figure 4: Profile dialog

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.

2.4   Width and Height

With these two dialogs you can define the limb's cross sections by specifying width and height distribution along the profile curve.

Figure 5: Width dialog

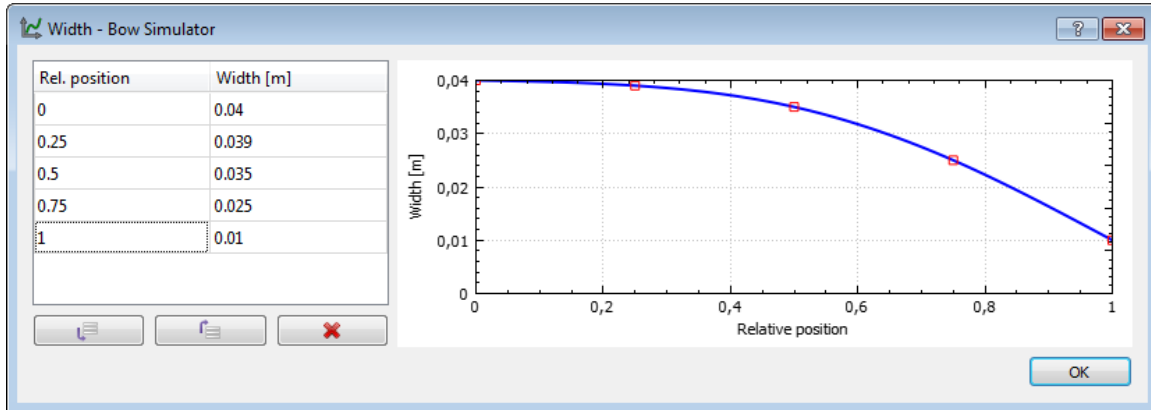Use the table on the left to specify width or height values at certain relative positions along the limb. A smooth curve (cubic spline) passing through the supplied values is constructed, which you can see on the plot.

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

Example: If you have a profile curve with a total length of 0.8m and you define three relative positions as 0, 50 and 100 respectively then the first cross section is placed at arc length 0m (beginning), the second at 0.4m and the third at 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.

2.5   Material

Here you can specify the properties of the limb material.
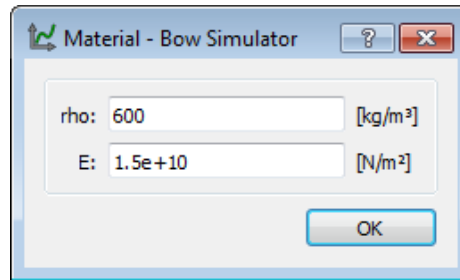
Figure 6: Material dialog

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 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 perform a material test. See section 5.3 for a simple bending test.

2.6 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 7: 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)
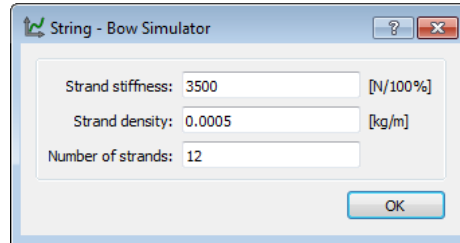- **Number of strands:** Total number of strands in the string

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 made by Alan Case.

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

Material	Stiffness $[\text{N}/100\%]$	Density $[\text{kg}/\text{m}]$	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

2.7 Masses

These are used to account for the various dead weights that can be attached to a bow.
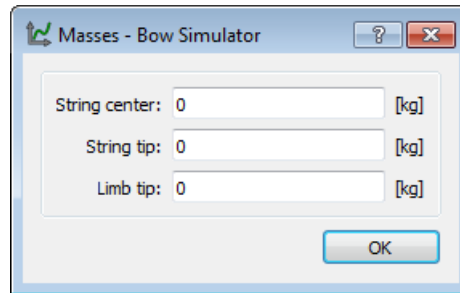
Figure 8: Masses dialog

- **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)

2.8 Operation

Here you can find all parameters that don't define the bow itself but rather the conditions under which it operates.
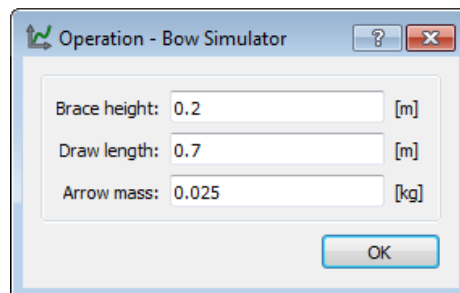
Figure 9: Operation dialog

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

3   Simulation Results

When you're done with editing the bow model you can run the simulation using either the yellow and green toolbar buttons or the simulation menu. Once the calculations are finished, a new window with the results opens.
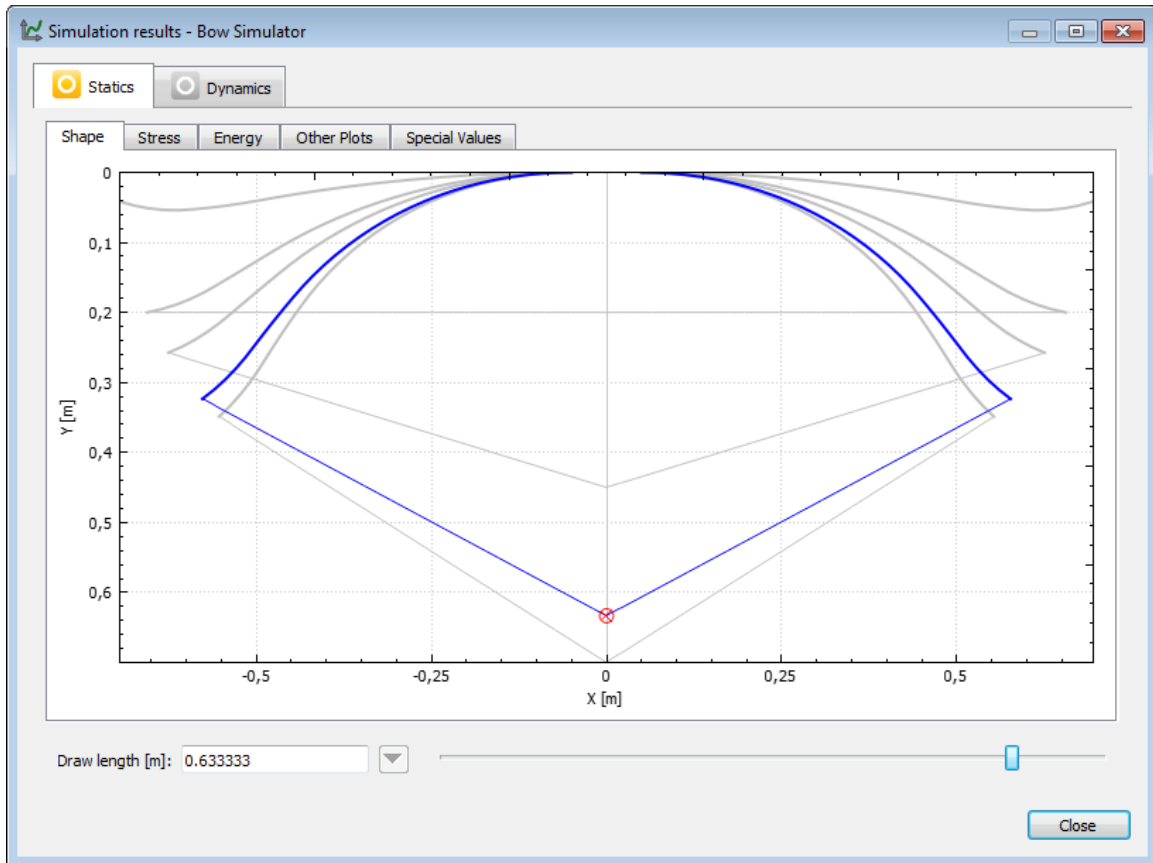
Figure 10: Simulation results

At the top of the result window you can switch between the static and dynamic results (if available).

At the bottom of the result 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. The result tabs are:

- **Shape:** This 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 arbitrary simulation results and plot them together.

- **Special values:** Most of those characterize some aspect of the bow's performance. They are different for the static and dynamic case.

  - **Statics:**

    - **String length:** Length of the string such that the bow meets the brace height specified by the user

    - **Final draw force:** Draw force in fully drawn state

    - **Drawing work:** The 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

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

    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:** Final velocity of the arrow when leaving the bow

    - **Arrow energy:** Final kinetic energy of the arrow when leaving the bow

    - **Efficiency:** Degree of efficiency of the bow. This is the ratio between energy input (static drawing work) and useful energy output (kinetic energy of the arrow)

4   Command Line Interface

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

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

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 parameters are optional. Calling Bow Simulator with either no options 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 and the results are written to the output file. If no output file was specified a default one with the same name as the input file is created.

Input and Output Formats

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

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

Both JSON and MessagePack are very common formats with implementations available in many programming languages. An example on how Bow Simulator can be used from Python is shown in Appendix C.

---

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

5 Background Information

5.1 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.)

5.2   Validation of Simulation Results

A very important task 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.

5.2.1   Statics of a straight steel bow

In this experiment the draw curve and limb shapes of a small steel bow have been measured and compared to version 2014.4 of Bow Simulation Tool (now Bow Simulator). The bow is shown in figure 11 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 11: 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 12 shows a comparison between the measured and the simulated draw curve and figure 13 compares the pictures of the limb against the simulated limb shapes.
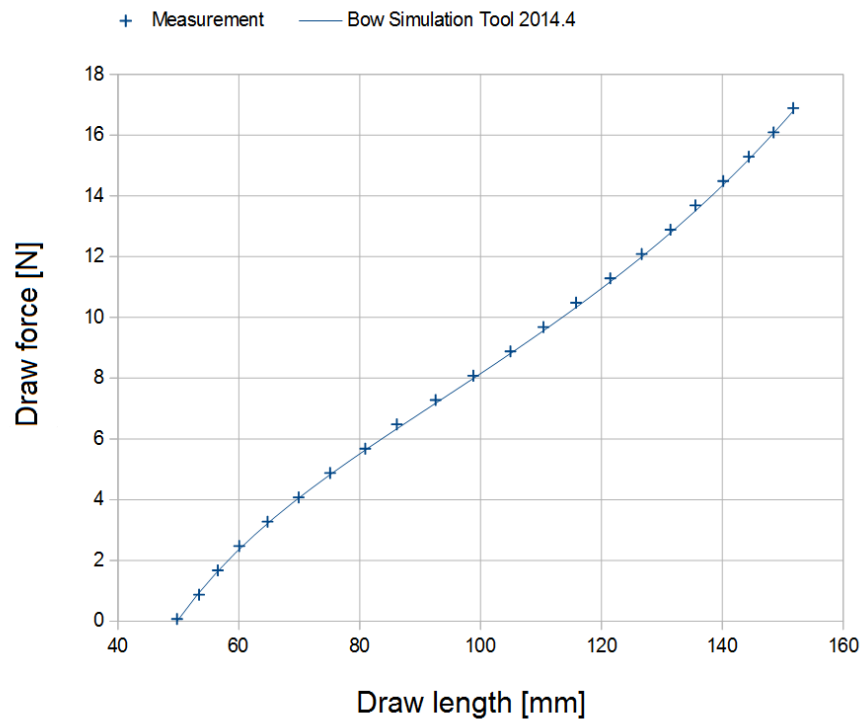
Figure 12: Experimental and simulated draw curves
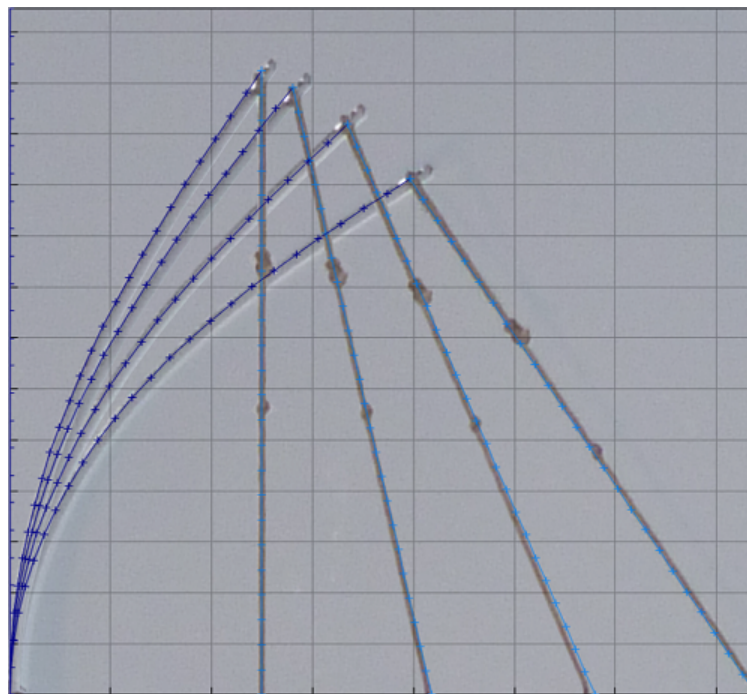
Figure 13: Experimental and simulated limb shapes

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.

5.3 A Simple Bending Test

A bending test is an easy way to determine the elastic modulus of a material because it can be done without any special equipment. In the test shown here a ledge with length L is clamped on one side and subjected to a vertical force F at the free end. The deflection s due to this load is measured. The elastic modulus can then be calculated using the equations in Table 2.
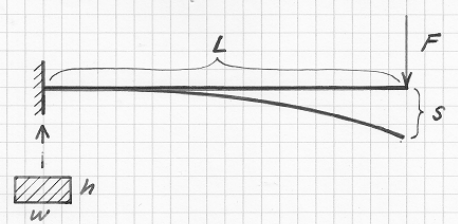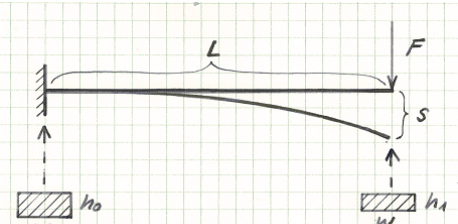
Constant cross sections	
	$E = \dfrac{4}{wh^3}\dfrac{FL^3}{s}$
Linearly tapered height	
	$E = \dfrac{12\ln(h_1 L)+6}{w\,(h_1-h_0)^3}\dfrac{FL^3}{s}$

Table 2: Calculation of the 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 long, slender beams and small deflections. The test setup should be chosen accordingly.

- 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.

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 in static analysis
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 of the dynamic output
profile			
segments			
args	double[]	m	Array of segment lengths
vals	double[]	1/m	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
E	double	Pa	Elastic modulus
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 (static or dynamic)
`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	Initial string length (unstressed)
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_force	`double[N]`	N	Draw force
draw_length	`double[N]`	m	Draw length
y_arrow	`double[N]`	m	Y coordinates of the arrow
x_limb	`double[N][P]`	m	X coordinate of the limb nodes
y_limb	`double[N][P]`	m	Y coordinates 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 (measured in -Y direction from full draw)
vel_arrow	`double[N]`	m/s	Arrow velocity (measured in -Y direction)
acc_arrow	`double[N]`	m/s^2	Arrow acceleration (measured in -Y direction)
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 example 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 Simulator input files (`.bow`) 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

# Open 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")

# Open 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/