Goal oriented self-adaptive development of online assignments based on autonomous computational methods

Andrew Pownuk[1], Iwona Skalna[2], M.V. Rama Rao[3]

1 - University of Texas at El Paso, El Paso, Texas, USA
2 - AGH University of Science and Technology, Krakow, Poland
3 - Vasavi College of Engineering, Hyderabad, India

28th Joint UTEP/NMSU Workshop on Mathematics, Computer Science, and Computational Sciences
El Paso, Texas, November 5, 2022

Outline

Online Learning

Figure: Blackboard

WebAssign

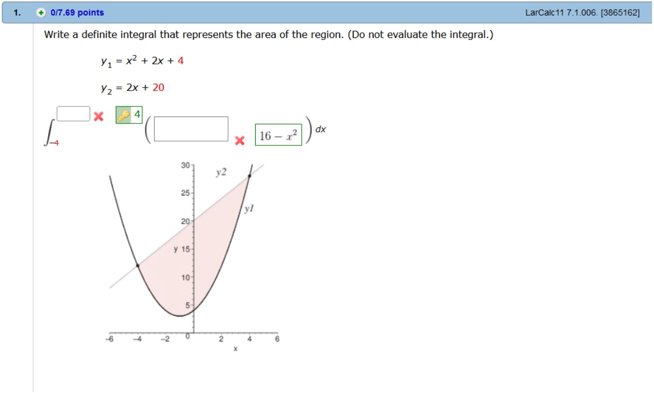


Figure: Sample assignment

Moodle

Figure: Online learning platform

WebWork

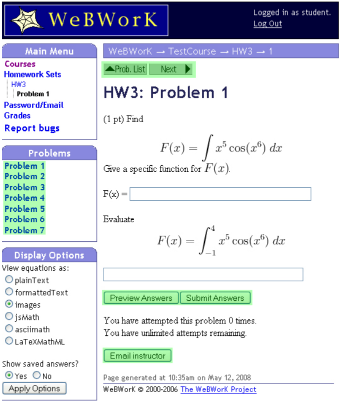
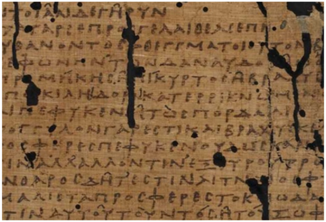
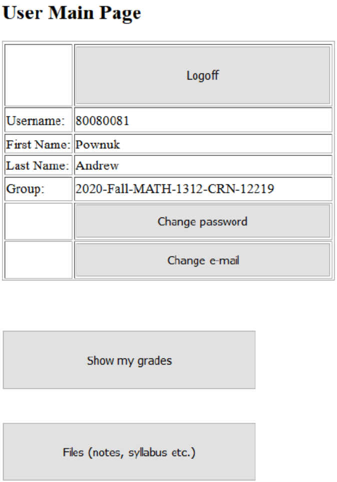


Figure: Sample assignment

Early Examples of Distance Learning



Online Learning



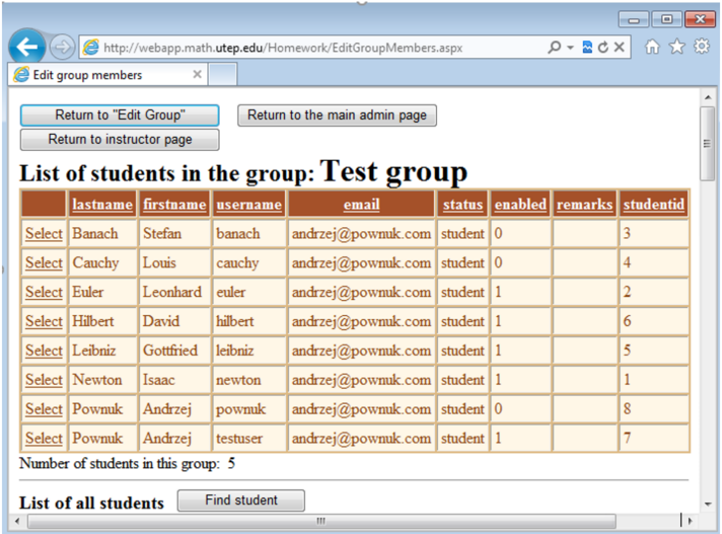
User Main Page

	Logoff
Username:	80080081
First Name:	Pownuk
Last Name:	Andrew
Group:	2020-Fall-MATH-1312-CRN-12219
	Change password
	Change e-mail

Show my grades

Files (notes, syllabus etc.)

List of Students


http://webapp.math.utep.edu/Homework/EditGroupMembers.aspx
Edit group members

Return to "Edit Group"
Return to the main admin page
Return to instructor page

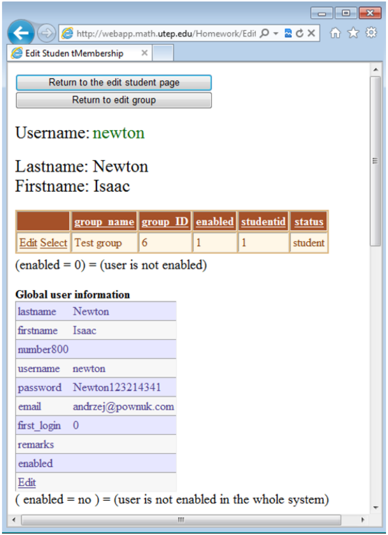
List of students in the group: Test group

	lastname	firstname	username	email	status	enabled	remarks	studentid
Select	Banach	Stefan	banach	andrzej@pownuk.com	student	0		3
Select	Cauchy	Louis	cauchy	andrzej@pownuk.com	student	0		4
Select	Euler	Leonhard	euler	andrzej@pownuk.com	student	1		2
Select	Hilbert	David	hilbert	andrzej@pownuk.com	student	1		6
Select	Leibniz	Gottfried	leibniz	andrzej@pownuk.com	student	1		5
Select	Newton	Isaac	newton	andrzej@pownuk.com	student	1		1
Select	Pownuk	Andrzej	pownuk	andrzej@pownuk.com	student	0		8
Select	Pownuk	Andrzej	testuser	andrzej@pownuk.com	student	1		7

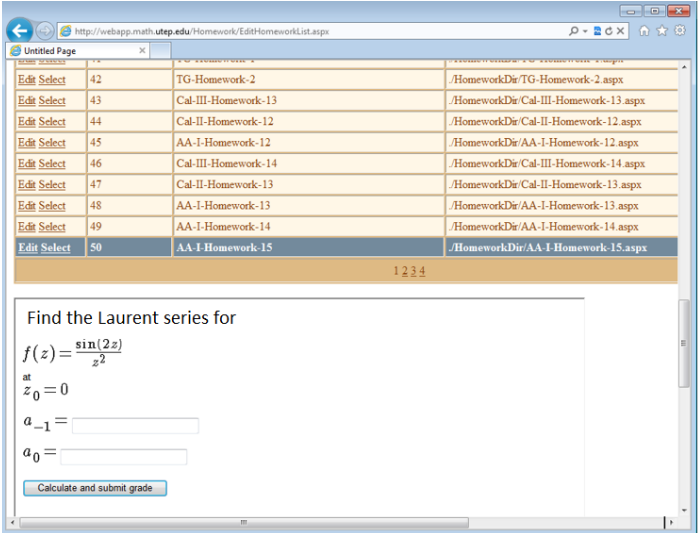
Number of students in this group: 5

List of all students Find student

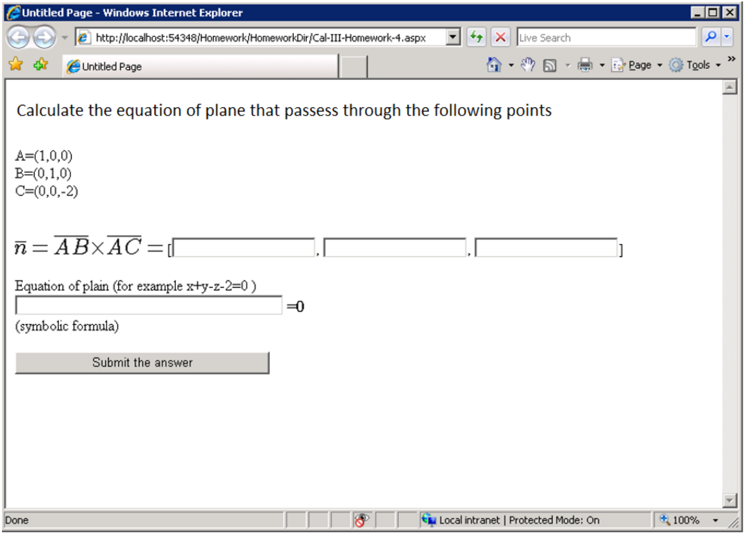
Information About Particular Student



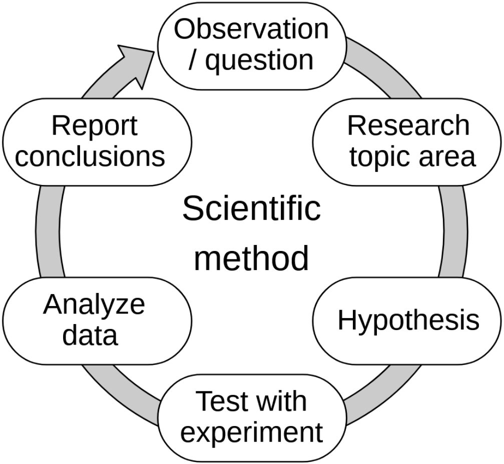
Online Homework



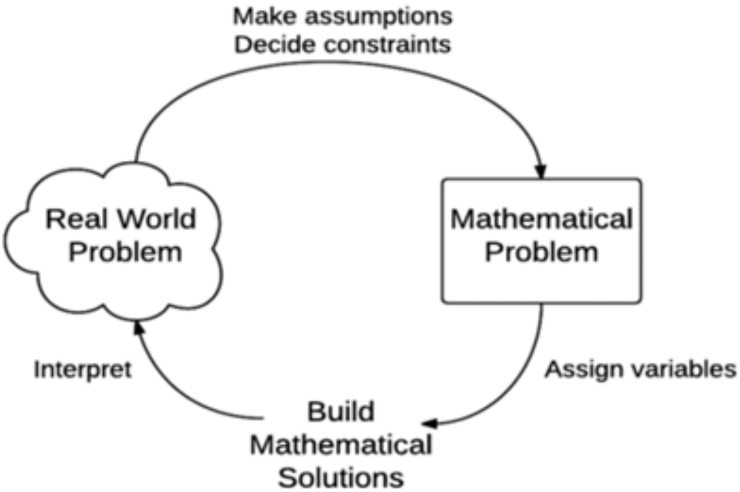
Online Homework



Scientific Method (Aristotle 384–322 BCE)



Mathematical Modeling

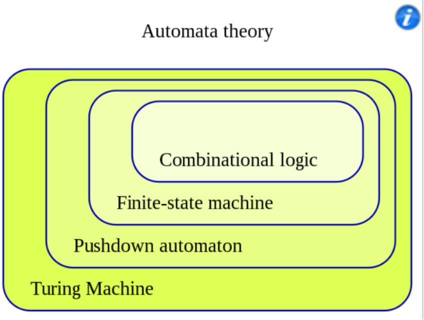


Curry–Howard Correspondence

The Curry–Howard correspondence (also known as the Curry–Howard isomorphism or equivalence, or the proofs-as-programs and propositions- or formulate-as-types interpretation) is the direct relationship between computer programs and mathematical proofs.
Teacher/scientists can use this relation and create computer programs from mathematical formulations and mathematical formulations from computer programs.

Turing Machine

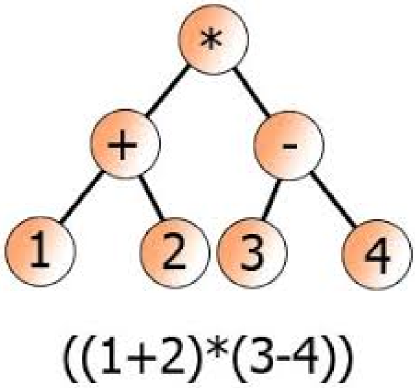
A Turing machine is a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, it is capable of implementing any computer algorithm.



Turing Complete Programming Languages in Teaching

Any Turing complete programming language (e.g. C,C++,Python, Java etc.) can express wide class of algorithms.
This gives teachers a tool to work interactively online on problems with practically arbitrary complexity.
To use this tool, it is necessary to represent a mathematical problem in a form of computer code and then the students by using webpages can interact with this code to check their progress. In the same way it is possible to work with research problems with arbitrary complexity.

Expression Tree



How to Evaluate Mathematical Expression Given as a String?

Example

- Expression: sqrt(2)*6
- Value: $\sqrt{2} \cdot 6$

Example

- Expression: sqrt2*6
- Value: $\sqrt{2} \cdot 6$

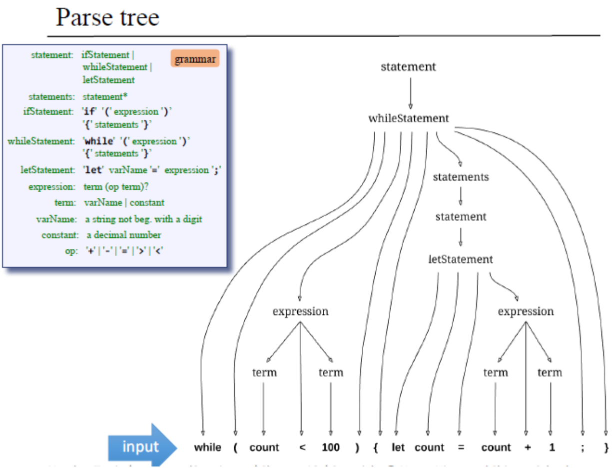
Example

- Expression: Sqrt[2]6
- Value: $\sqrt{2} \cdot 6$

Example

- Expression: SQRT[2]6
- Value: $\sqrt{2} \cdot 6$

Parse Tree



Parse tree
grammar
statement: ifStatement | whileStatement | letStatement
statements: statement*
ifStatement: 'if' '(' expression ')' '{' statements '}'
whileStatement: 'while' '(' expression ')' '{' statements '}'
letStatement: 'let' varName '=' expression ';'
expression: term (op term)?
term: varName | constant
varName: a string not beg. with a digit
constant: a decimal number
op: '+' | '-' | '=' | '>' | '<'
statement
whileStatement
statements
statement
letStatement
expression
term
term
expression
term
term
input
while (count < 100) { let count = count + 1 ; }

XML Parse Tree

Parse tree

grammar

```
statement:      ifStatement |
                whileStatement |
                letStatement
statements:     statement*
ifStatement:    'if' '(' expression ')'
                '{' statements '}'
whileStatement: 'while' '(' expression ')'
                '{' statements '}'
letStatement:   'let' varName '=' expression ';'
expression:     term (op term)?
term:           varName | constant
varName:        a string not beg. with a digit
constant:       a decimal number
op:             '+' | '-' | '=' | '>' | '<'
```

parser output

```
<whileStatement>
  <keyword> while </keyword>
  <symbol> ( </symbol>
  <expression>
    <term>
      <identifier> count </identifier>
    </term>
    <symbol> < </symbol>
    <term>
      <intConstant> 100 </intConstant>
    </term>
  </expression>
  <symbol> ) </symbol>
  <symbol> { </symbol>
  <statements>
    <letStatement>
      <keyword> let </keyword>
      <identifier> count </identifier>
      <symbol> = </symbol>
      <expression>
        <term> <identifier> count </identifier> </term>
        <symbol> + </symbol>
        <term> <intConstant> 1 </intConstant> </term>
      </expression>
      <symbol> ; </symbol>
    </letStatement>
  </statements>
  <symbol> } </symbol>
</whileStatement>
```

Same parse tree, in XML

Grammar

<table>
<tr><td>Lexical elements:</td><td>The Jack language includes five types of terminal elements (tokens):</td></tr>
<tr><td>keyword:</td><td>'class' | 'constructor' | 'function' | 'method' | 'field' | 'static' |
'var' | 'int' | 'char' | 'boolean' | 'void' | 'true' | 'false' | 'null' | 'this' |
'let' | 'do' | 'if' | 'else' | 'while' | 'return'</td></tr>
<tr><td>symbol:</td><td>'{' | '}' | '(' | ')' | '[' | ']' | '.' | ',' | ';' | '+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '=' | '~'</td></tr>
<tr><td>integerConstant:</td><td>A decimal number in the range 0 .. 32767.</td></tr>
<tr><td>StringConstant</td><td>'"' A sequence of Unicode characters not including double quote or newline '"'</td></tr>
<tr><td>identifier:</td><td>A sequence of letters, digits, and underscore ('_') not starting with a digit.</td></tr>
<tr><td>Program structure:</td><td>A Jack program is a collection of classes, each appearing in a separate file.
The compilation unit is a class. A class is a sequence of tokens structured according to the following context free syntax:</td></tr>
<tr><td>class:</td><td>'class' className '{' classVarDec* subroutineDec* '}'</td></tr>
<tr><td>classVarDec:</td><td>('static' | 'field') type varName (',' varName)* ';'</td></tr>
<tr><td>type:</td><td>'int' | 'char' | 'boolean' | className</td></tr>
<tr><td>subroutineDec:</td><td>('constructor' | 'function' | 'method') ('void' | type) subroutineName
'(' parameterList ')' subroutineBody</td></tr>
<tr><td>parameterList:</td><td>((type varName) (',' type varName)*)?</td></tr>
<tr><td>subroutineBody:</td><td>'{' varDec* statements '}'</td></tr>
<tr><td>varDec:</td><td>'var' type varName (',' varName)* ';'</td></tr>
<tr><td>className:</td><td>identifier</td></tr>
<tr><td>subroutineName:</td><td>identifier</td></tr>
<tr><td>varName:</td><td>identifier</td></tr>
<tr><td>Statements:</td><td></td></tr>
<tr><td>statements:</td><td>statement*</td></tr>
<tr><td>statement:</td><td>letStatement | ifStatement | whileStatement | doStatement | returnStatement</td></tr>
<tr><td>letStatement:</td><td>'let' varName ('[' expression ']')? '=' expression ';'</td></tr>
<tr><td>ifStatement:</td><td>'if' '(' expression ')' '{' statements '}' ('else' '{' statements '}')?</td></tr>
<tr><td>whileStatement:</td><td>'while' '(' expression ')' '{' statements '}'</td></tr>
<tr><td>doStatement:</td><td>'do' subroutineCall ';'</td></tr>
<tr><td>ReturnStatement</td><td>'return' expression? ';'</td></tr>
<tr><td>Expressions:</td><td></td></tr>
<tr><td>expression:</td><td>term (op term)*</td></tr>
<tr><td>term:</td><td>integerConstant | stringConstant | keywordConstant | varName |
varName '[' expression ']' | subroutineCall | '(' expression ')' | unaryOp term</td></tr>
<tr><td>subroutineCall:</td><td>subroutineName '(' expressionList ')' | (className | varName) '.' subroutineName
'(' expressionList ')'</td></tr>
<tr><td>expressionList:</td><td>(expression (',' expression)*)?</td></tr>
<tr><td>op:</td><td>'+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='</td></tr>
<tr><td>unaryOp:</td><td>'-' | '~'</td></tr>
<tr><td>KeywordConstant:</td><td>'true' | 'false' | 'null' | 'this'</td></tr>
</table>

Infix Notation, Prefix Notation, Postfix Notation

Different notation for arithmetic expressions.

- Infix notation $(5+6)x7$
- Prefix notation $x+567$
- Postfix notation $756+x$

Typical evaluation process of arithmetic expressions.

- InfixToPrefix($(5+6)x7$)$=x+567$
- EvaluatePrefix($x+567$)$=77$

WebAssign Question Modes

Mode	Description
Algebraic	Students submit a mathematical expression or equation that is evaluated algebraically.
Essay	Students submit an extended textual response. Scored manually.
File-Upload	Students submit a file. Scored manually.
Fill-in-the-Blank	Students submit a brief textual response.
Graphing	Students draw on a Cartesian coordinate plane.
Image Map	Students click a displayed image.
Java	Students use Java, Flash, or other applet to answer the question.
Matching	Students match items from two lists.
Multiple-Choice	Students select one response from a list.
Multiple-Select	Students select one or more responses from a list.
NumberLine	Students place or draw points, lines, segments, or rays.
Numerical	Students submit a numerical response which might include units or be checked for significant figures.
Poll	Used with other modes to create questions that collect information only. All responses are scored correct.
Symbolic	Students submit a mathematical expression (not an equation) that is evaluated by value substitution.

WebAssign Questions from Code (HTML, Perl etc.)

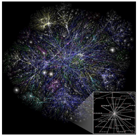
The following code in Answer accepts two values between 20 and 100, and then requires your students to multiply them:

```
<EQN $A=userinput(20,100,50)>
<EQN $B=userinput(20,100,50)>
<EQN $SIMPLIFIED=1; $A * $B>
```

To require students to perform the computations instead of stating their response as a mathematical expression, set the $SIMPLIFIED, $FRACTION, or $PROPERFRACTION variables to 1 as you would for any other numerical question.

TCP/IP

The Internet protocol suite, commonly known as TCP/IP, is a framework of organizing the set of communication protocols used in the Internet and similar computer networks according to functional criteria. The foundational protocols in the suite are the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP), and the Internet Protocol (IP). In the development of this networking model, early versions of it were known as the Department of Defense (DoD) model because the research and development were funded by the United States Department of Defense through DARPA. ARPANET started in 1969.



HTML

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Initial release 1993, Tim Berners-Lee, CERN (European Organization for Nuclear Research).

PHP

Paradigm	Multi-paradigm: imperative, functional, object-oriented, procedural, reflective
Designed by	Rasmus Lerdorf
Developer	The PHP Development Team, Zend Technologies, PHP Foundation
First appeared	June 8, 1995; 27 years ago[1][2]
Stable release	8.1.12 / 2022-10-25
Preview release	8.2.0 beta / 2022-07-21
Typing discipline	Dynamic, weak, gradual[3]
Implementation language	C (primarily; some components C++)
OS	Unix-like, Windows, macOS, IBM i, OpenVMS
License	PHP License (most of Zend engine

JavaScript

Paradigm	Multi-paradigm: event-driven, functional, imperative, procedural, object-oriented programming
Designed by	Brendan Eich of Netscape initially; others have also contributed to the ECMAScript standard
First appeared	December 4, 1995; 26 years ago[1]
Stable release	ECMAScript 2021[2] / June 2021; 16 months ago
Preview release	ECMAScript 2022[3] / 22 July 2021; 15 months ago
Typing discipline	Dynamic, weak, duck
Filename extensions	`.js` · `.cjs` · `.mjs` [4]

Asp.Net

Developer(s)	Microsoft
Initial release	January 5, 2002; 20 years ago
Stable release	6 / February 8, 2022; 8 months ago[1]
Written in	.NET languages
Operating system	Microsoft Windows, Linux, macOS
Platform	Cross-platform
Type	Web framework
License	Apache License 2.0
Website	www.asp.net

Automated Development of Mathematical Description

- General formulation of the problem.
- Formulation of possible special cases.
- Construction of the search space for a given problem.
- Finding possible limitations and stopping criteria.

$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+\cos(\cos(x))) \tag{708}$$
$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+(2+2)) \tag{709}$$
$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+(2+x)) \tag{710}$$
$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+(2+\sin(x))) \tag{711}$$
$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+(2+\cos(x))) \tag{712}$$
$$\frac{d}{dx}\cos(x)+\frac{d}{dx}(2+(x+x)) \tag{713}$$

Automated Development of Step-by-step Solution

step-1

$((1+2)+(2+1))=x$

step-2

$((1+2)+(2+1))=x$

step-3

$(3+(2+1))=x$

step-4

$((2+1)+1+2)=x$

step-5

$((1+2)+3)=x$

step-6

$((1+2)+2+1)=x$

step-7

$(3+(2+1))=x$

step-8

$((2+1)+1+2)=x$

step-9

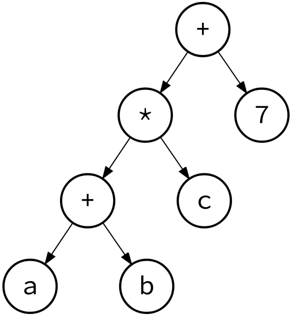
$((1+2)+3)=x$

step-10

$((1+2)+2+1)=x$

Conversion to Computer Code

- Mathematical formulation can be represented by a computational graphs.
- Graphs can be translated into computer code in given programming language.



Uploading Code to the Server


```
#!/bin/bash
HOST="hostname"
USER="username"
PASSWORD="password"
DESTINATION=$1
NEWDIR=$2
FILE=$3
ftp -inv $HOST <<EOF
user $USER $PASSWORD
cd $DESTINATION
pwd
mkdir $NEWDIR
cd $NEWDIR
pwd
passive
mput *.aspx
```

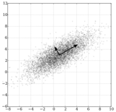
Goal Oriented Self-adaptive Development of Online Assignments

- In some cases it is possible to construct very specific computational graph which is related to given problem.
- Computational graph for a given problem can be represented in many different ways.
- Family of computational graphs can be translated into specific computer code.
- Computer code can be embedded into existing online learning framework and formulated as an assignment for student.

Example - Finding Eigenvalues

- Generate a matrix A with known eigenvalues.
- Write characteristic equation.
  $a_n\lambda^n + a_{n-1}\lambda^{n-1} + ... + a_1 x + a_0 = 0$
- Find eigenvalues $\lambda_1, ..., \lambda_n$.
- Find related eigenvectors and generalized eigenvectors $(v_1, v_2, ..., v_n)$ and construct related Jordan form of the matrix $J = P^{-1}AP$.
- Apply the primary decomposition theorem
  $S = N((A - \lambda_1 I)^{k_1}) \oplus ... \oplus N((A - \lambda_m I)^{k_m})$

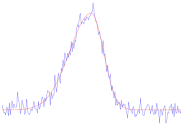
Presented problem has many special cases and related assignments require appropriate software which controls steps of the calculations.



Example - Discrete Least Square Approximation

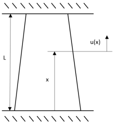
- Specify the model $\phi(x, w)$ $(w = (w_1, w_2, ..., w_m))$.
- Specify the loss function $L(x, y, \phi(x, w))$ for given discrete set of values $x = (x_1, ..., x_n), y = (y_1, ..., y_n)$.
- Specify an algorithm for finding $w = \arg\min_w L(x, y, \phi(x, w))$
- Find parameters w and specify the model $\phi(x, w)$.

In presented assignment it is possible to consider various loss functions and models ϕ (polynomial approximation, neural networks, etc.) and optimization algoritms (direct optimization,gradient decent, stochastic gradient descent etc.).



Example - Weak Formulation and FEM Method

For the following boundary value problem
$\frac{d}{dx}\left(EA\frac{du}{dx}\right) + n = 0, u(0) = 0, u(L) = 0$
Find related variational equation
$a(u, v) = l(v) \forall v \in H_0^1(\Omega)$
After calculations we know that
$a(u, v) = \int_0^L EA\frac{du}{dx}\frac{dv}{dx}dx, l(v) = \int_0^L nvdx$
$\|u\|_{W^{k,p}(\Omega)} = \left(\sum_{|\alpha|\leqslant k} \|D^\alpha u\|_{L^p(\Omega)}^p\right)^{\frac{1}{p}}$
In presented assignment it is possible to consider various approximation spaces $V_h \subset H_0^1(\Omega)$, geometry of the problem, and loads. The Lax-Milgram theorem can be applied to prove existence and uniquness of the solution.



Numerical Symulations in Linear/Nonlinear Elasticity

Equation of motion

$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \mathbf{F} = \rho \ddot{\mathbf{u}}$

Constitutive equations

$\boldsymbol{\sigma} = \mathsf{C} : \boldsymbol{\varepsilon}$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\boldsymbol{\varepsilon}$ is the infinitesimal strain tensor, $\mathbf{u}$ is the displacement (vector), C is the fourth-order stiffness tensor, $\mathbf{F}$ is the body force per unit volume, ρ is the mass density.

$C_{ijkl} = K\,\delta_{ij}\,\delta_{kl} + \mu\,(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk} - \tfrac{2}{3}\,\delta_{ij}\,\delta_{kl}$

$\sigma_{ij} = K\delta_{ij}\varepsilon_{kk} + 2\mu\left(\varepsilon_{ij} - \tfrac{1}{3}\delta_{ij}\varepsilon_{kk}\right)$

$\varepsilon_{rs} = \frac{1}{2}\left(\frac{\partial u_r}{\partial x_s} + \frac{\partial u_s}{\partial x_r} + \frac{\partial u_k}{\partial x_r}\frac{\partial u_k}{\partial x_s}\right) \approx \frac{1}{2}\left(\frac{\partial u_r}{\partial x_s} + \frac{\partial u_s}{\partial x_r}\right)$

In presented assignment it is possible to consider diffrent geometry of the problems, loads, shape functions, approximation spaces etc.

Conclusions

- Turing complete programming languages give teachers a tool to work interactively online on problems with practically arbitrary complexity.
- By using Turing complete computer languages it is possible to implement very wide class of mathematical problems by using computer code.
- Many engineering problems can be describe by using mathematical models, because of that it is possible to generate automatically online assignments for students.
- It is continent to create a set of goal-oriented assignments.
- In some cases every step of generation of online learning assignments can be fully automated and speed up by using autonomous computational algorithms.

Thank You